An Open-Source Implementation to Predict Buckling Behaviour of Cold-Formed Sections

S-J Bronkhorst

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering at Stellenbosch University

Supervisor: Dr G.C. van Rooyen

Faculty of Engineering

Department of Civil Engineering

March 2017

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2017

Abstract

The deformation and buckling behaviour of cold-formed sections is a popular research topic. It is currently being studied at Stellenbosch University, but only a few tools exist to aid such research of which only one, namely CUFSM [18], is open-source.

In this thesis, Finite Strip theory is reviewed to give the reader sufficient understanding in order to develop an implementation. A new, object-oriented Finite Strip Method (FSM) implementation is created to demonstrate how the FSM can be used for static analysis of members as well as to predict their buckling behaviour. This implementation is then further expanded to include the ability to do Direct Strength based designs directly from the strength curves it calculates. The implementation is tested and the results are compared to existing FSM and FEM implementations. The results delivered by the implementation were found to be similar to those of CUFSM in terms of buckling analysis and similar to those of FEM in terms of static analysis.

A brief overview of the Direct Strength and Effective Width Methods is provided. The design methods are compared by looking at three aspects namely: design effort, accuracy, and economy. The Finite Strip Method is discussed with emphasis on why this method is favoured as an input for the Direct Strength Method.

Abstrak

Die deformasie en knikgedrag van koud-gevormde staal profiele is 'n populêre navorsings onderwerp. Hierdie onderwerp word tans aan die Universiteit van Stellenbosch ondersoek, maar navorsers het 'n berperkte aantal sagteware toepassings tot hulle beskikking, waarvan slegs een, naamlik CUFSM oop bron is.

In hierdie tesis word die Eindige Strook teorie hersien om die leser voldoende kennis te bied om hy/sy eie toepassing te ontwikkel. 'n Nuwe objek georiënteerde Eindige Strook implementering word geskep om te demonstreer hoe die Eindige Strook Metode gebruik kan word vir die statiese analise sowel as om die knik gedrag van balke en kolomme te voorspel. Hierdie implementering word dan verder uitgebrei om die vermoë te hê om Direkte Krag gebasseerde ontwerpe te doen direk vanaf die krag kurwes wat dit genereer. Die toepassing word getoets en die resultate word vergelyk met bestaande Eindige Strook en Eindige Element (EEM) implementerings. Die tesis vind dat die resultate gelewer deur die toepassing soortgelyk is aan daardie van CUFSM in die geval van knik gedrag en soortgelyk aan die EEM in die geval van statiese analise.

'n Vlugtige oorsig van die Direkte Krag en Effektiewe Wydte metodes word voorsien. Hierdie twee

ontwerp metodes word vergelyk op grond van drie aspekte naamlik: ontwerp moeite, akuraatheid en ekonomie. Die Eindige Strook Metode word bespreek met klem op hoekom hierdie metode verkies word as 'n inset tot die Direkte Krag Metode.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Cold-formed steel sections are widely used in lightweight steel construction. It may serve as efficient primary framing systems in low to mid-rise construction and secondary framing systems in high-rise construction. It is also used in speciality structures such as storage racks and greenhouses [2]. The cold-forming process enables manufacturers to create various section types. The resulting members are prismatic, typically not doubly-symmetric and consist of slender plates. A consequence of having members that consist of plates having a high width to thickness ratio is that stability of the cross section must be considered in their design because it influences their behaviour under load [11]. The lack of symmetry in many of these cross sections, as well as other unique characteristics, introduce complexity when creating a simple design method. Specifically, local buckling of the plates comprising a cold-formed cross-section and cross-section distortion are essential parts of member design. For successful design, it is necessary to understand the complex stability behaviour of these members and account for it.

Research in the field of cold-formed members usually includes the analysis of these members by use of the finite element method or similar methods. For everyday analysis, the finite element method is the most commonly used structural analysis tool. It is powerful, well-known, versatile and well established. However, for the analysis of structures with regular geometry and simple boundary conditions, a finite element analysis is somewhat extravagant and may be time-consuming.

Another method exists, that assumes regular geometry and simple boundary conditions in a more simplified, computationally efficient model while sacrificing minimal accuracy. This method is called the finite strip method (FSM) and it has been partially implemented by only a few analysis programs. In the past, FSM has been used in the design and vibration analysis of bridge decks. The direct strength method, which is gaining popularity in design codes, has the requirement that buckling modes must be separated. In recent times FSM has been used as an input to the direct strength method because of its ability to conveniently separate buckling modes.

1.2 Motivation

The deformation and buckling behaviour of cold-formed sections is a popular research topic. It is currently being studied at Stellenbosch University, but only a few tools (Table 1) exist to aid such research of which only one is open-source.

Table 1: Available FSM implementations.

Tool	**License type**
CUFSM	Open-source (MATLAB^{TM})
CFS	Paid
THIN-WALL	Paid
GBTUL[1]	Freeware

CUFSM is the most widely used cold-formed member design tool, mainly because it is the only open-source implementation available. However, CUFSM is not a complete implementation and only focusses on solving stability problems using FSM. CUFSM only demonstrates how the FSM is used to perform buckling analysis problems, it does not demonstrate how a static analysis is performed using the FSM. Furthermore, CUFSM is implemented in MATLAB^{TM}. The consequence of this being that the use of CUFSM requires MATLAB^{TM} or the MATLAB^{TM} runtime environment. In order for a researcher to view or modify CUFSM's source code, they would first need to purchase a copy of MATLAB^{TM} since the MATLAB^{TM} runtime environment does not provide this functionality.

It therefore makes sense to create a fundamental implementation of FSM, in a programming language that is more accessible, in order to make FSM technology itself practicable and understandable. An implementation adhering to this criteria would be a valuable research tool in order to assist in the development of design procedures, which can further be developed into design tools.

Currently one of the main uses of the Finite Strip Method is it to predict the buckling behavior of cold-formed sections of various lengths. This output is then represented as a buckling curve or signature curve. Local minima are read off from the signature and used as input into the Direct Strength Method. Usually, the Direct Strength design is done by some other software or by hand. It would make sense to incorporate this functionality into a Finite Strip implementation in order to eliminate the need for third-party software or hand calculations. This will, in turn, save money on costs for software and time which would have been used for hand calculations. Furthermore it could eliminate any chance for errors.

[1]GBTUL is listed here because it performs the same task as the FSM implementations, but relies on Generalized Beam Theory rather than FSM.

1.3 Aims and Objectives

In this thesis, the aim will be to create a Finite Strip software model that demonstrates the fundamental theory of the Finite Strip Method. The program source code should be understandable for persons having a basic programming background in order to enable researchers to easily view and modify the program to suit their needs. The program should have the ability to do Direct Strength calculations from the buckling data it computes and present this data in a manner that is easy to interpret.

The intermediate objectives towards reaching the desired outcomes are as follows:

1. Review the fundamental Finite Strip theory to gain sufficient understanding in order to develop an implementation.
2. Develop a Finite Strip implementation in JAVA that demonstrates the fundamental theory. The implementations source code should be understandable and easy to modify.
3. Test the fundamental implementation to ensure that the output is reliable, and may be used in design methods. Compare the results to existing FSM and FEM implementations.
4. Expand the fundamental implementation to include Direct Strength design.
5. Test the implementation to gain confidence in the designs it produces. Compare the results to examples in design manuals.

2 Numerical modelling of elastic buckling analysis

The individual plates that make up a cold-formed section typically have high slenderness ratios. This results in a member that buckles at stresses less than yield stress if the load causes part of the member to be in compression. All design specifications (e.g. SANS 10162-2) investigate elastic buckling based on rational analysis, then use empirical equations that are calibrated to experimental data to predict member strength [14] [6].

The prediction of elastic buckling is crucial to the design procedure. Many numerical analysis methods are available for the prediction of elastic buckling such as the Finite Element Method (FEM) [17], Finite Strip Method (FSM) [1], constrained Finite Strip Method (cFSM) [18] [2] and Generalized Beam Theory (GBT) [19].

Each of the above-mentioned methods provides a means to calculate the buckling load of a member given a certain length. The typical buckling-load versus member length curve or buckling curve for cold-formed sections can be classified into three buckling modes: local, distortional and global or Euler buckling. Each buckling mode exhibits different post-buckling behaviour and some modes may be coupled with or influenced by others.

2.1 The Finite Element Method (FEM)

For everyday analysis, the FEM is the most commonly used structural analysis tool. It is powerful, well-known, versatile and well established. Shell finite elements have been increasingly used in the analysis of cold-formed members [12]. The ability to handle a range of boundary conditions, consider moment gradient, account for shear effects and handle members with varying cross-sections along their length, make the FEM appealing. However, for the analysis of structures with regular geometry and simple boundary conditions, a Finite Element analysis is somewhat extravagant and may be excessively time-consuming in pre-processing and analysis time as well as post-processing time.

Finite element analysis using thin plates or shell elements may be used for elastic buckling prediction, but there are two reasons to avoid it. Firstly, the number of elements required for reasonable accuracy can be significant, thus increasing analysis time. Secondly, interpretation of the output is hard because the method does not seperate the buckling modes.

Figure 1 highlights the importance of separating buckling modes.

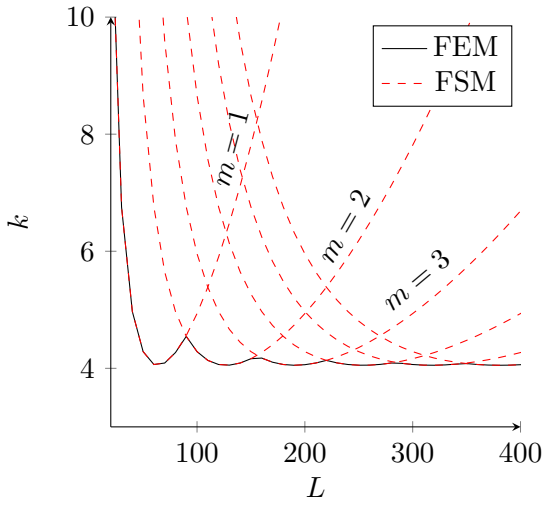


Figure 1: Plate buckling coefficient, mixed vs. separated modes.

The various curves resulting from an FSM analysis each represent the minimum load required for buckling to occur for different half-wave shapes. The first of these curves is of particular interest and is typically referred to as the signature curve, the buckling load versus half-wave length curve. From Figure 1 it is clear that the output from a typical FEM analysis does not provide any information on how much a specific buckling mode contributes to the overall strength of the member. The FEM output is in the form of a buckling curve with mixed modes. It provides us with the minimum buckling load for all modes of buckling, but not one mode in particular. Therefore the local minima corresponding to local, distortional and global buckling can not be uniquely identified. This makes the output unfavourable for use as input into the DSM.

2.2 The Finite Strip Method (FSM)

The Finite Strip Method (FSM) can be considered as a specialization of the Finite Element Method (FEM) [20]. The two methods share the same basic methodology and theory. The displacement field is defined as a combination of shape functions in terms of nodal degrees of freedom etc. The main difference is in the way FSM discretizes a member. The displacement function is approximated by simple polynomials in the transverse direction, but in the longitudinal direction, FSM utilises continuously differentiable smooth series functions that approximate the displacement function and satisfy the boundary conditions at the ends of the strip *a priori*, while FEM would use simple polynomials in all directions. This has the consequence that in FSM a prismatic three-dimensional structure, in other words having a uniform cross section, is modelled as a two-dimensional structure.

Finite Strip analysis is one of the most efficient and popular methods for the elastic buckling prediction of thin-walled sections. In the past, FSM has also been used in the design and vibration analysis of bridge decks [1]. The Direct Strength Method (DSM), which is gaining popularity in design codes, has the requirement that buckling modes must be separated. In recent times, FSM is used as input to the DSM because of its ability to conveniently separate buckling modes.

The efficiency of the FSM is due the fact that it does not discretize members along their lengths but instead employs specially selected shape functions in the longitudinal direction that satisfy the boundary conditions at the ends of the strip *a priori*. The implication being that fewer elements are used to describe a member than one would normally use in the general method. This causes a significant decrease in the number of degrees of freedom that need to be solved.

For the simply supported case, the FSM is problem is inherently separated into a number of parallel tasks of which the solutions are independent of one another. For example, a buckling curve with 100 plot points can theoretically be solved in the same time as one plot point if enough CPU cores are available. Similarly, for static analysis of simply supported members higher accuracy may be achieved by increasing the number of longitudinal terms used. For each chosen term a separate system of equations has to be solved. This process can also be done in parallel to increase performance.

The FSM provides accurate elastic buckling solutions with minimal effort and time. However, the FSM has two major limitations. Firstly, the FSM assumes simply supported boundary conditions at the member ends. In a paper by Li [2] it is stated that for all boundary conditions other than simply supported, an interaction of buckling modes of different half-wavelengths occur and the half wavelength vs. buckling load curve loses its special significance. In these cases, FSM has the same buckling mode identification problem as FEM, unless other tools such as the constrained Finite Strip Method (cFSM) are implemented [2]. Secondly, since the FSM does not discretize members along their length, it only supports members that are prismatic.

However, for the design of cold formed members the above mentioned limitations do not pose a threat to FSM since these members are in fact prismatic, and most designs are based on the assumption that the members are indeed simply-supported.

3 Numerical Model - Finite Strip Method

In the Finite Strip Method (FSM), the cross section of a structure is approximated. A network of nodes is defined on the cross section. These nodes become nodal lines when viewing the structure in 3D. In this way, the structure is divided into two-dimensional sub-domains (strips) in which one opposite pair of sides coincide with the boundaries of the structure. The geometry of the structure is usually constant along its length so that the width of the strip does not change from one end to the other.

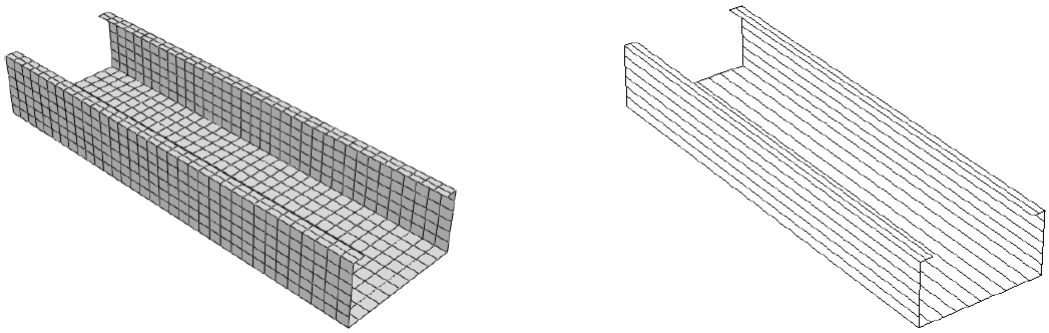


Figure 2: Typical FEM mesh vs. FSM mesh for thin-wall lipped channel (Zhanjie Li [2])

Since it is a specialized form of FEM, Cheung [1] states that the philosophy of the FSM and FEM are similar. They both require the descritization of the continuum so that only a finite number of unknowns will exist in the resulting formulation. This procedure can be described as follows: (i) The continuum is divided into strips by imaginary lines called nodal lines. The ends of the strips always form a part of the continuum's boundary. (ii) The strips are assumed to be connected to one another along a number of nodal lines which coincide with the longitudinal boundaries of the strip. The degrees of freedom at each nodal line, called nodal displacement parameters, are related to the displacements and their first partial derivatives with respect to the polynomial variable, x, in the transverse direction. (iii) A displacement function, in terms of the nodal displacement parameters, is chosen to represent the displacement field and consequently the strain and stress fields within each strip. (iv) Based on the chosen displacement function, it is possible to obtain a stiffness matrix and load matrices which balance the various loads acting on the strip through either virtual work or energy prinicples. (v) The stiffness and load matrices of all the strips are assembled to form a set of overall stiffness equations. The equations can be solved by any standard matrix solution technique to yield the nodal displacement parameters.

3.1 Displacement Functions, Shape Functions and Degrees of Freedom

The general form of the displacement function, w, is given as a product of a polynomial function, $f_m(x)$, and a series function, $Y_m(y)$.

$$w = \sum_{m=1}^{q} f_m(x) Y_m \tag{1}$$

From the previous discussion it is clear that choosing a suitable displacement function for a strip is the most crucial part of the analysis. An incorrectly chosen displacement function might not only lead to incorrect results, but could lead to results that converge to the wrong answer for successively refined meshes. Cheung [1] provides a lengthy discussion on how to ensure displacement functions are chosen correctly. Since a displacement function always consists of two parts, it would be convenient to discuss each part separately.

3.1.1 Series part of displacement function

Since the longitudinal displacement function is interpolated by longitudinal shape functions, the choice of suitable interpolation functions for a strip play a key role in the application of FSM. There are several functions available to use as longitudinal shape functions that have all been shown to be effective. According to Cheung [1], the most commonly used series are the basic functions which are derived from the solution of the beam vibration differential equation

$$Y'''' = \frac{\mu^4}{a^4} Y, \tag{2}$$

where a is the length of the strip and μ is a parameter.

The general form of the basic functions is

$$Y(y) = C_1 \sin\frac{\mu y}{a} + C_2 \cos\frac{\mu y}{a} + C_3 \sinh\frac{\mu y}{a} + C_4 \cosh\frac{\mu y}{a} \tag{3}$$

with the coefficients C_1 etc., to be determined by the end conditions. These have been calculated explicitly in the literature for various end conditions. Some are listed below:

(a) Both ends simply supported ($Y(0) = Y''(0) = 0, Y(a) = Y''(a) = 0$).

$$Y_m(y) = \sin\frac{\mu_m y}{a} \qquad (\mu_m = \pi, 2\pi, 3\pi, ...m\pi) \tag{4}$$

(b) Both ends clamped ($Y(0) = Y'(0) = 0, Y(a) = Y'(a) = 0$).

$$\begin{aligned} Y_m(y) &= \sin\frac{\mu_m y}{a} - \sinh\frac{\mu_m y}{a} - \alpha_m\left[\cos\frac{\mu_m y}{a} - \cosh\frac{\mu_m y}{a}\right] \\ \alpha_m &= \frac{\sin\mu_m - \sinh\mu_m}{\cos\mu_m - \cosh\mu_m} \\ (\mu_m &= 4.7300, 7.8532, 10.9960, \ldots, \frac{2m+1}{2}\pi) \end{aligned} \tag{5}$$

(c) One end clamped and the other end free ($Y(0) = Y'(0) = 0, Y''(a) = Y'''(a) = 0$).

$$\begin{aligned} Y_m(y) &= \sin\frac{\mu_m y}{a} - \sinh\frac{\mu_m y}{a} - \alpha_m\left[\cos\frac{\mu_m y}{a} - \cosh\frac{\mu_m y}{a}\right] \\ \alpha_m &= \frac{\sin\mu_m + \sinh\mu_m}{\cos\mu_m + \cosh\mu_m} \\ (\mu_m &= 1.875, 4.694, \ldots, \frac{2m-1}{2}\pi) \end{aligned} \tag{6}$$

Note that all of the above series functions satisfy the longitudinal boundary conditions of the strip *a priori*. For example, for a simply supported strip in bending, the displacement function is able to satisfy the conditions of both deflection, w, which can be considered as a displacement condition and bending moment $\frac{\partial^2 w}{\partial y^2}$, which can be considered as a force condition at the ends of the strip. CUFSM utilizes different series functions from the ones stated here, namely trigonometric functions [2]. The trigonometric functions used by CUFSM only satisfy the displacement boundary conditions, not the force boundary conditions, and consequently can only be used for vibration or buckling problems and not to perform static analysis [1] [8]. However, all the harmonic functions used here, except for the function corresponding to simply-supported boundary conditions, have the problem that their products with each of their derivatives can only be integrated numerically, which makes them slow to use in a computer programme whereas the functions utilized by CUFSM can be integrated analytically.

3.1.2 Polynomial Part of Displacement Function

A shape function is a polynomial associated with a nodal displacement parameter. It describes the corresponding displacement field within the cross-section of a strip when the nodal displacement parameter in question is given a unit value. In fact, such shape functions are derived by specifying a normalized unit value of the relevant displacement component at its own node, and a value of zero for the same displacement component at all the other nodes.

For example, Equation 1 may be written as

$$w = \sum_{m=1}^{q} [[C_1][C_2]] \begin{Bmatrix} \{\delta_1\} \\ \{\delta_2\} \end{Bmatrix}_m Y_m \tag{7}$$

in which $\begin{Bmatrix} \{\delta_1\} \\ \{\delta_2\} \end{Bmatrix}_m$ is a vector representing the mth term nodal displacement parameters, deflection and rotation, at nodes 1 and 2, and $[C_1]$ and $[C_2]$ are the shape functions associated with $\{\delta_1\}$ and $\{\delta_2\}$ respectively.

Many shape functions are available. According to Cheung [1] the most common ones, which will also be used in this thesis, are:

(a) For a strip with two nodal lines and displacements as nodal parameters:

$$C_1 = (1 - \bar{x}), C_2 = \bar{x} \tag{8}$$

(b) For a strip with two nodal lines and with displacements and their first derivatives as nodal parameters:

$$[C_1] = [(1 - 3\bar{x}^2 + 2\bar{x}^3), x(1 - 2\bar{x} + \bar{x}^2)] \qquad \text{and} \qquad [C_2] = [(3\bar{x}^2 - 2\bar{x}^3), x(\bar{x}^2 - \bar{x})] \tag{9}$$

With $\bar{x} = x/b$

$b =$ the width of the strip.

3.1.3 Plate Strip

As in FEM, numerous types of finite strip elements have been developed for different usages. For example, the curved plate strip for modelling structures that are curved in plan. This thesis will focus on the lower order rectangular plate strip, which is a combination of the two-dimensional bending strip and the two-dimensional plane strip proposed by Cheung [1]. A typical lower order plate strip is shown in Figure 3 along with the degrees of freedom (u, v, w, θ) on the node lines and the local (x, y, z) coordinate system. Note that the degrees of freedom are not at a specific point on the nodal lines but rather form a "field" along them.

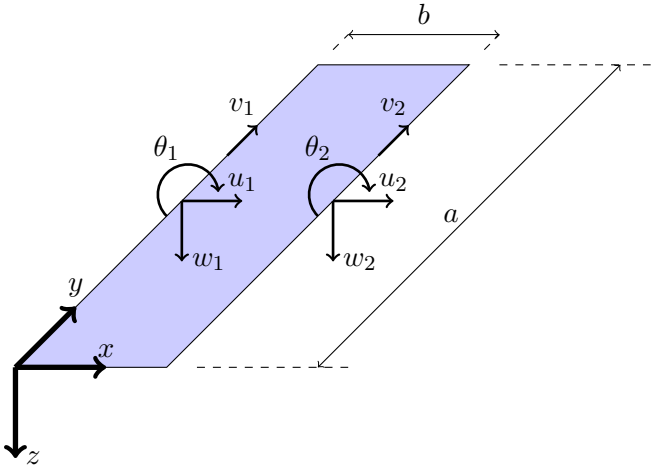


Figure 3: Coordinates and degrees of freedom of a strip.

The plane strip and bending strip proposed by Cheung [1] use the shape functions given by equations 8 and 9 respectively. Since the plate strip is a combination of these two, it will employ the plane strip's shapefunctions to relate the plane nodal parameters u_1, u_2, v_1 and v_2 to the plane displacement functions u and v, and the bending strip's shape functions to relate the bending nodal parameters, w_1, w_2, θ_1 and θ_2 to the bending displacement function, w.

In matrix form the displacement fields can then be approximated with the shape functions, N, and nodal displacements d as follows:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \sum_{m=1}^{q} \left[N_{uv}^{[m]}\right] \{d_{uv}^{[m]}\} \qquad \text{and} \qquad \{w\} = \sum_{m=1}^{q} \left[N_{w}^{[m]}\right] \{d_{w}^{[m]}\} \tag{10}$$

Where $d_{uv}^{[m]}$ = the nodal parameters associated with plane behavior = $\left[u_{1[m]}\ v_{1[m]}\ u_{2[m]}\ v_{2[m]}\right]^T$, $d_{w}^{[m]}$ = the nodal parameters associated with bending behavior = $\left[w_{1[m]}\ \theta_{1[m]}\ w_{2[m]}\ \theta_{2[m]}\right]^T$, $\left[N_{uv}^{[m]}\right]$ = the shape functions associated with plane behavior and $\left[N_{w}^{[m]}\right]$ = the shape functions associated with bending behavior. The series function, $Y_{[m]}$, used for interpolation in the longitudinal direction will

consist of a total of q terms. This means that for each value $1, .., m, .., q$ there will exist a nodal parameter associated with term m. The interpolated values for u, v and w are calculated as the sum of the interpolated values for each m term up and until q, specifically:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \sum_{m=1}^{q} \begin{bmatrix} (1-\bar{x})Y_{[m]} & 0 & (\bar{x})Y_{[m]} & 0 \\ 0 & (1-\bar{x})\dfrac{a}{\mu_{[m]}}Y'_{[m]} & 0 & (\bar{x})\dfrac{a}{\mu_{[m]}}Y'_{[m]} \end{bmatrix} \begin{Bmatrix} u_{1[m]} \\ v_{1[m]} \\ u_{2[m]} \\ v_{2[m]} \end{Bmatrix} \tag{11}$$

Note that in the above formulation for the series part of the displacement field Y is used for u while Y' is used for v. This is based on the observation of the relationship commonly used in the small deflection theory of beams, in which the transverse deflection, u, is related to the longitudinal displacement, v, through:

$$v = A\frac{du}{dy} \tag{12}$$

The interpolated value for w is calculated as follows:

$$\{w\} = \sum_{m=1}^{q} Y_{[m]} \begin{bmatrix} (1-3\bar{x}^2+2\bar{x}^3) & x(1-2\bar{x}+\bar{x}^2) & (3\bar{x}^2-2\bar{x}^3) & x(\bar{x}^2-\bar{x}) \end{bmatrix} \begin{Bmatrix} w_{1[m]} \\ \theta_{1[m]} \\ w_{2[m]} \\ \theta_{2[m]} \end{Bmatrix} \tag{13}$$

With :

$\bar{x} = \dfrac{x}{b}$

a : The length of the strip as shown in Figure 3.

b : The width of the strip as shown in Figure 3.

$Y_{[m]}$: The term in the chosen series function corresponding to m.

μ_m : A parameter as defined by the chosen series function.

3.2 Stiffness Matrices

The stiffness matrices of the FSM are used in a similar manner to those of the FEM. For instance, element stiffness matrices are set up for each element and then assembled into the system matrix. The same goes for the element vectors. However, since FSM employs series functions as shape functions in the longitudinal direction, the element and system matrices consist of a set of submatrices, each corresponding to a term in the series function. In the case of simply supported boundary conditions, these terms are uncoupled and each submatrix may be formed and assembled similarly to the FEM and solved for each term as a seperate system of equations. For all other boundary conditions, the terms of the series function are coupled and the assembly process, which is explained later, becomes more complicated.

The elastic and geometric stiffness matrices for lower order rectangular plate elements are listed below. Their derivations are tiresome and well described in literature [1], consequently it is not repeated here. Each submatrix, $k_e^{[mn]}$, of the elastic stiffness matrix k_e can be divided into two parts, membrane stiffness $k_{eM}^{[mn]}$ and bending stiffness $k_{eB}^{[mn]}$, where m and n correspond to the longitudinal term numbers. The numbers in round brackets indicate the row and column in the corresponding matrix.

$$k_e^{[mn]} = \begin{bmatrix} k_{eM(1,1)}^{[mn]} & k_{eM(1,2)}^{[mn]} & \cdot & \cdot & k_{eM(1,3)}^{[mn]} & k_{eM(1,4)}^{[mn]} & \cdot & \cdot \\ k_{eM(2,1)}^{[mn]} & k_{eM(2,2)}^{[mn]} & \cdot & \cdot & k_{eM(2,3)}^{[mn]} & k_{eM(2,4)}^{[mn]} & \cdot & \cdot \\ \cdot & \cdot & k_{eB(1,1)}^{[mn]} & k_{eB(1,2)}^{[mn]} & \cdot & \cdot & k_{eB(1,3)}^{[mn]} & k_{eB(1,4)}^{[mn]} \\ \cdot & \cdot & k_{eB(2,1)}^{[mn]} & k_{eB(2,2)}^{[mn]} & \cdot & \cdot & k_{eB(2,3)}^{[mn]} & k_{eB(2,4)}^{[mn]} \\ k_{eM(3,1)}^{[mn]} & k_{eM(3,2)}^{[mn]} & \cdot & \cdot & k_{eM(3,3)}^{[mn]} & k_{eM(3,4)}^{[mn]} & \cdot & \cdot \\ k_{eM(4,1)}^{[mn]} & k_{eM(4,2)}^{[mn]} & \cdot & \cdot & k_{eM(4,3)}^{[mn]} & k_{eM(4,4)}^{[mn]} & \cdot & \cdot \\ \cdot & \cdot & k_{eB(3,1)}^{[mn]} & k_{eB(3,2)}^{[mn]} & \cdot & \cdot & k_{eB(3,3)}^{[mn]} & k_{eB(3,4)}^{[mn]} \\ \cdot & \cdot & k_{eB(4,1)}^{[mn]} & k_{eB(4,2)}^{[mn]} & \cdot & \cdot & k_{eB(4,3)}^{[mn]} & k_{eB(4,4)}^{[mn]} \end{bmatrix} \tag{14}$$

With :

$$k_{eM}^{[mn]} = t \begin{bmatrix} \left[\left(\frac{K_1}{b}\right)I_1 + \left(\frac{K_4 b}{3}\right)I_5\right] & \left[\left(\frac{-K_2}{2C_2}\right)I_3 + \left(\frac{-K_4}{2C_2}\right)I_5\right] & \left[\left(\frac{-K_1}{b}\right)I_1 + \left(\frac{K_4 b}{6}\right)I_5\right] & \left[\left(\frac{-K_2}{2C_2}\right)I_3 + \left(\frac{K_4}{2C_2}\right)I_5\right] \\ \left[\left(\frac{-K_2}{2C_1}\right)I_2 + \left(\frac{-K_4}{2C_1}\right)I_5\right] & \left[\left(\frac{K_3 b}{3C_1C_2}\right)I_4 + \left(\frac{K_4}{bC_1C_2}\right)I_5\right] & \left[\left(\frac{K_2}{2C_1}\right)I_2 + \left(\frac{-K_4}{2C_1}\right)I_5\right] & \left[\left(\frac{K_3 b}{6C_1C_2}\right)I_4 + \left(\frac{-K_4}{6C_1C_2}\right)I_5\right] \\ \left[\left(\frac{-K_1}{b}\right)I_1 + \left(\frac{K_4 b}{6}\right)I_5\right] & \left[\left(\frac{K_2}{2C_2}\right)I_3 + \left(\frac{-K_4}{2C_2}\right)I_5\right] & \left[\left(\frac{K_1}{b}\right)I_1 + \left(\frac{K_4 b}{3}\right)I_5\right] & \left[\left(\frac{K_2}{2C_2}\right)I_3 + \left(\frac{K_4}{2C_2}\right)I_5\right] \\ \left[\left(\frac{-K_2}{2C_1}\right)I_2 + \left(\frac{K_4}{2C_1}\right)I_5\right] & \left[\left(\frac{K_3 b}{6C_1C_2}\right)I_4 + \left(\frac{-K_4}{bC_1C_2}\right)I_5\right] & \left[\left(\frac{K_2}{2C_1}\right)I_2 + \left(\frac{K_4}{2C_1}\right)I_5\right] & \left[\left(\frac{K_3 b}{3C_1C_2}\right)I_4 + \left(\frac{K_4}{bC_1C_2}\right)I_5\right] \end{bmatrix} \tag{15}$$

and

$$k_{eB}^{[mn]} = \frac{1}{420b^3}\cdot$$

$$\left[\begin{array}{cccc}
\begin{pmatrix} 5040D_xI_1 - 504b^2D_1I_2 \\ -504b^2D_1I_3 + 156b^4D_yI_4 \\ +2016b^2D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} 2520bD_xI_1 - 462b^3D_1I_2 \\ -42b^3D_1I_3 + 22b^5D_yI_4 \\ +168b^3D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} -5040D_xI_1 + 504b^2D_1I_2 \\ +504b^2D_1I_3 + 54b^4D_yI_4 \\ -2016b^2D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} 2520D_xI_1 - 42b^3D_1I_2 \\ -42b^3D_1I_3 - 13b^5D_yI_4 \\ +168b^3D_{xy}I_5 \end{pmatrix} \\
\begin{pmatrix} 2520bD_xI_1 - 462b^3D_1I_3 \\ -42b^3D_1I_2 + 22b^5D_yI_4 \\ +168b^3D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} 1680b^2D_xI_1 - 56b^4D_1I_2 \\ -56b^4D_1I_3 + 4b^6D_yI_4 \\ +224b^4D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} -2520bD_xI_1 + 42b^3D_1I_2 \\ +42b^3D_1I_3 + 13b^5D_yI_4 \\ -168b^3D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} 840b^2D_xI_1 + 14b^4D_1I_2 \\ +14b^4D_1I_3 - 3b^6D_yI_4 \\ -56b^4D_{xy}I_5 \end{pmatrix} \\
\begin{pmatrix} -5040D_xI_1 + 504b^2D_1I_2 \\ +504b^2D_1I_3 + 54b^4D_yI_4 \\ -2016b^2D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} -2520bD_xI_1 + 42b^3D_1I_2 \\ +42b^3D_1I_3 + 13b^5D_yI_4 \\ -168b^3D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} 5040D_xI_1 - 504b^2D_1I_2 \\ -504b^2D_1I_3 + 156b^4D_yI_4 \\ +2016b^2D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} -2520bD_xI_1 + 462b^3D_1I_2 \\ +42b^3D_1I_3 - 22b^5D_yI_4 \\ -168b^3D_{xy}I_5 \end{pmatrix} \\
\begin{pmatrix} 2520bD_xI_1 - 42b^3D_1I_2 \\ -42b^3D_1I_3 - 13b^5D_yI_4 \\ +168b^3D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} 840b^2D_xI_1 + 14b^4D_1I_2 \\ +14b^4D_1I_3 - 3b^6D_yI_4 \\ -56b^4D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} -2520bD_xI_1 + 462b^3D_1I_3 \\ +42b^3D_1I_2 - 22b^5D_yI_4 \\ -168b^3D_{xy}I_5 \end{pmatrix} & \begin{pmatrix} 1680b^2D_xI_1 - 56b^4D_1I_2 \\ -56b^4D_1I_3 + 4b^6D_yI_4 \\ +224b^4D_{xy}I_5 \end{pmatrix}
\end{array}\right] \tag{16}$$

where: $I_1 = \int_0^a Y_m Y_n \, dy$, $I_2 = \int_0^a Y_m'' Y_n \, dy$, $I_3 = \int_0^a Y_m Y_n'' \, dy$, $I_4 = \int_0^a Y_m'' Y_n'' \, dy$, $I_5 = \int_0^a Y_m' Y_n' \, dy$,

$K_1 = \dfrac{E_x}{1 - v_x v_y}$, $K_2 = \dfrac{v_x E_y}{1 - v_x v_y}$, $K_3 = \dfrac{E_y}{1 - v_x v_y}$, $K_4 = G_{xy}$, $C_1 = \dfrac{\mu_m}{a}$, $C_2 = \dfrac{\mu_n}{a}$, $D_x = \dfrac{E_x t^3}{12(1 - v_x v_y)}$,

$D_y = \dfrac{E_y t^3}{12(1 - v_x v_y)}$, $D_1 = \dfrac{v_y E_x t^3}{12(1 - v_x v_y)}$, $D_{xy} = \dfrac{G t^3}{12}$.

Note that $k_{eM}^{[mn]}$ and $k_{eB}^{[mn]}$ are in general non-symmetric, also the integral numbering convention of Li [2] is followed instead of the convention used by Cheung [1]. The full elastic stiffness matrix k_e , which is in fact symmetric, can be expressed as:

$$k_e = \left[k_e^{[mn]}\right]_{q \times q} \tag{17}$$

Since a plate strip has four degrees of freedom at each node, in the case of the two node plate strip discussed here each $k_e^{[mn]}$ sub-matrix has dimensions 8×8 and q^2 such sub-matrices exist. Where q is the total number of longitudinal terms chosen such that $m = 1, 2, ..., q$ and $n = 1, 2, ..., q$. For the case where both longitudinal boundary conditions are simply-supported (S-S), I_1 through I_5 are zero when $m \neq n$, leaving only the diagonal set of sub-matrices in k_e. This important property makes FSM efficient at identifying and separating buckling modes. For all other boundary conditions, FSM has the same identification problems as FEM unless another technique such as the constrained Finite Strip Method (cFSM) is employed [2].

Similar to the elastic stiffness matrix, the geometric stiffness matrix $k_g^{[mn]}$ corresponding to longitudinal term numbers m and n, is divided into a membrane part $k_{gM}^{[mn]}$ and a bending part $k_{gB}^{[mn]}$.

$$k_g^{[mn]} = \begin{bmatrix} k_{gM(1,1)}^{[mn]} & k_{gM(1,2)}^{[mn]} & \cdot & \cdot & k_{gM(1,3)}^{[mn]} & k_{gM(1,4)}^{[mn]} & \cdot & \cdot \\ k_{gM(2,1)}^{[mn]} & k_{gM(2,2)}^{[mn]} & \cdot & \cdot & k_{gM(2,3)}^{[mn]} & k_{gM(2,4)}^{[mn]} & \cdot & \cdot \\ \cdot & \cdot & k_{gB(1,1)}^{[mn]} & k_{gB(1,2)}^{[mn]} & \cdot & \cdot & k_{gB(1,3)}^{[mn]} & k_{gB(1,4)}^{[mn]} \\ \cdot & \cdot & k_{gB(2,1)}^{[mn]} & k_{gB(2,2)}^{[mn]} & \cdot & \cdot & k_{gB(2,3)}^{[mn]} & k_{gB(2,4)}^{[mn]} \\ k_{gM(3,1)}^{[mn]} & k_{gM(3,2)}^{[mn]} & \cdot & \cdot & k_{gM(3,3)}^{[mn]} & k_{gM(3,4)}^{[mn]} & \cdot & \cdot \\ k_{gM(4,1)}^{[mn]} & k_{gM(4,2)}^{[mn]} & \cdot & \cdot & k_{gM(4,3)}^{[mn]} & k_{gM(4,4)}^{[mn]} & \cdot & \cdot \\ \cdot & \cdot & k_{gB(3,1)}^{[mn]} & k_{gB(3,2)}^{[mn]} & \cdot & \cdot & k_{gB(3,3)}^{[mn]} & k_{gB(3,4)}^{[mn]} \\ \cdot & \cdot & k_{gB(4,1)}^{[mn]} & k_{gB(4,2)}^{[mn]} & \cdot & \cdot & k_{gB(4,3)}^{[mn]} & k_{gB(4,4)}^{[mn]} \end{bmatrix} \tag{18}$$

With:

$$k_{gM}^{[mn]} = \begin{bmatrix} \frac{(3T_1+T_2)bI_5}{12} & \cdot & \frac{(T_1+T_2)bI_5}{12} & \cdot \\ & \frac{(3T_1+T_2)ba^2I_4}{12\mu_m\mu_n} & \cdot & \frac{(T_1+T_2)ba^2I_4}{12\mu_m\mu_n} \\ & & \frac{(T_1+3T_2)bI_5}{12} & \cdot \\ \text{symmetric} & & & \frac{(T_1+3T_2)ba^2I_4}{12\mu_m\mu_n} \end{bmatrix} \tag{19}$$

and

$$k_{gB}^{[mn]} = \begin{bmatrix} \frac{(10T_1+3T_2)bI_5}{35} & \frac{(15T_1+7T_2)b^2I_5}{420} & \frac{9(T_1+T_2)bI_5}{140} & -\frac{(7T_1+6T_2)b^2I_5}{420} \\ & \frac{(5T_1+3T_2)b^3I_5}{840} & \frac{(6T_1+7T_2)b^2I_5}{420} & -\frac{(T_1+T_2)b^3I_5}{280} \\ & & \frac{3T_1+10T_2)bI_5}{35} & -\frac{(7T_1+15T_2)b^2I_5}{420} \\ \text{symmetric} & & & \frac{(3T_1+5T_2)b^3I_5}{840} \end{bmatrix} \tag{20}$$

where: μ_m and μ_n are as given by the longitudinal displacement function Eq. 4 to 6. $I_4 = \int_0^a Y_m'' Y_n'' \, dy$, $I_5 = \int_0^a Y_m' Y_n' \, dy$.

For elastic buckling problems, a compressive edge traction is applied in the longitudinal direction causing a reduction in stiffness. The T_1 and T_2 terms in the geometric stiffness matrix are in fact the values of the distributed load at the nodes, not the stress. They are computed from the nodal stress by multiplying the stress at the node with the thickness of the strip e.g $T_1 = f_1 \times t$.

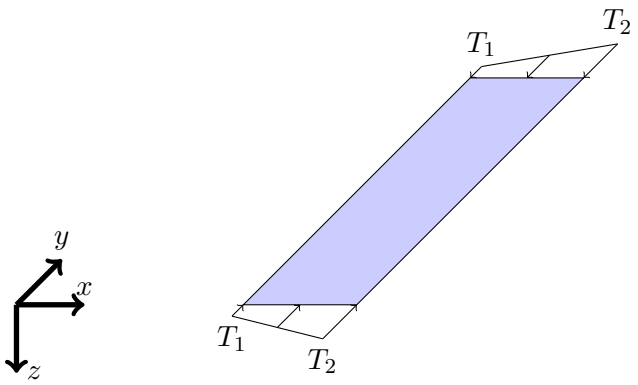


Figure 4: End traction applied to a strip.

The geometric stiffness matrix k_g can be expressed in its full form as:

$$k_g = \left[k_g^{[mn]}\right]_{q \times q} \tag{21}$$

Where q is the total number of longitudinal terms chosen such that $m = 1, 2, ..., q$ and $n = 1, 2, ..., q$. Each submatrix has dimensions 8×8 and q^2 such submatrices exist. The assembly process will be described in Section 3.6.

3.3 Loads and Edge Tractions

Derivation of the load vectors are tiresome and well described in literature [1], consequently it is not repeated here. Further, in order to create a software implementation of FSM closed form equations for the load vectors are all that are required since it is more computationally efficient to implement their solutions in closed form. The load vector $F^{[m]}$ corresponding to longitudinal term number m of a strip can be divided into a membrane and a bending part similar to the stiffness matrices.

$$\{F^{[m]}\} = \begin{Bmatrix} F_M^{[m]}{}_{(1)} \\ F_M^{[m]}{}_{(2)} \\ F_B^{[m]}{}_{(1)} \\ F_B^{[m]}{}_{(2)} \\ F_M^{[m]}{}_{(3)} \\ F_M^{[m]}{}_{(4)} \\ F_B^{[m]}{}_{(3)} \\ F_B^{[m]}{}_{(4)} \end{Bmatrix} \tag{22}$$

The load vectors for bending, $F_B^{[m]}$, and membrane behavior, $F_M^{[m]}$, are given by Cheung [1] and are calculated as follows:

$$\{F_B^{[m]}\} = \begin{Bmatrix} F_B^{[m]}{}_{(1)} \\ F_B^{[m]}{}_{(2)} \\ F_B^{[m]}{}_{(3)} \\ F_B^{[m]}{}_{(4)} \end{Bmatrix} = q_z \frac{b}{2} \begin{Bmatrix} 1 \\ \frac{b}{6} \\ 1 \\ \frac{-b}{6} \end{Bmatrix} \int_0^a Y_m \, dy \tag{23}$$

$$\{F_M^{[m]}\} = \begin{Bmatrix} F_M^{[m]}{}_{(1)} \\ F_M^{[m]}{}_{(2)} \\ F_M^{[m]}{}_{(3)} \\ F_M^{[m]}{}_{(4)} \end{Bmatrix} = \frac{b}{2} \begin{Bmatrix} q_x \int_0^a Y_m \, dy \\ q_y \frac{a}{\mu_y} \int_0^a Y'_m \, dy \\ q_x \int_0^a Y_m \, dy \\ q_y \frac{a}{\mu_y} \int_0^a Y'_m \, dy \end{Bmatrix} \tag{24}$$

where q_x , q_y and q_z are the constant area loads in the respective local x, y and z directions.

3.4 Strains and Stresses

It is simple to obtain the strains by differentiation of the displacement functions. For plate bending, the strains are given by the second partial derivative of the displacement function w. For membrane behaviour, the two direct strains ϵ_x and ϵ_y are given by the first derivatives of u and v respectively.

$$\{\epsilon_B\} = \begin{Bmatrix} -\chi_x \\ -\chi_y \\ 2\chi_{xy} \end{Bmatrix} = \begin{Bmatrix} -\dfrac{\partial^2 w}{\partial x^2} \\ -\dfrac{\partial^2 w}{\partial y^2} \\ \dfrac{2\partial^2 w}{\partial x \partial y} \end{Bmatrix} = \sum_{m=1}^{q} \begin{Bmatrix} -\dfrac{\partial^2 \left[N_w^{[m]}\right]}{\partial x^2} \\ -\dfrac{\partial^2 \left[N_w^{[m]}\right]}{\partial y^2} \\ \dfrac{2\partial^2 \left[N_w^{[m]}\right]}{\partial x \partial y} \end{Bmatrix} \{d_w^{[m]}\} = \sum_{m=1}^{q} [B_B^{[m]}]\{d_w^{[m]}\} \tag{25}$$

$$\{\epsilon_M\} = \begin{Bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{Bmatrix} = \sum_{m=1}^{q} [B_M^{[m]}]\{d_{uv}^{[m]}\} \tag{26}$$

The bending and membrane strain matrices are obtained by performing the appropriate differentiation of the interpolated displacements. Their explicit forms are as follows:

$$[B_B^{[m]}] = \begin{bmatrix} \dfrac{6}{b^2}(1-2\bar{x})Y_m & \dfrac{2}{b}(2-3\bar{x})Y_m & \dfrac{6}{b^2}(-1+2\bar{x})Y_m & \dfrac{2}{b}(-3\bar{x}+1)Y_m \\ -(1-3\bar{x}^2+2\bar{x}^3)Y_m'' & -x(1-2\bar{x}+\bar{x}^2)Y_m'' & -(3\bar{x}^2-2\bar{x}^3)Y_m'' & -x(\bar{x}^2-\bar{x})Y_m'' \\ \dfrac{2}{b}(-6\bar{x}+6\bar{x}^2)Y_m' & 2(1-4\bar{x}+3\bar{x}^2)Y_m' & \dfrac{2}{b}(6\bar{x}-6\bar{x}^2)Y_m' & 2(3\bar{x}^2-2\bar{x})Y_m' \end{bmatrix} \tag{27}$$

$$[B_M^{[m]}] = \begin{bmatrix} \frac{-1}{b}Y_m & 0 & \frac{1}{b}Y_m & 0 \\ 0 & (1-\bar{x})\frac{a}{\mu_m}Y_m'' & 0 & (\bar{x})\frac{a}{\mu_m}Y_m'' \\ (1-\bar{x})Y_m' & \frac{-1}{b}\frac{a}{\mu_m}Y_m' & (\bar{x})Y_m' & \frac{1}{b}\frac{a}{\mu_m}Y_m' \end{bmatrix} \tag{28}$$

The stresses caused by bending are given in the form of moments, while the stresses caused by membrane behaviour are given as regular stresses. Both are related to the strains through the bending and membrane material properties of the strip respectively.

$$\{\sigma_B\} = \begin{bmatrix} M_x \\ M_y \\ M_x y \end{bmatrix} = [D_B]\{\epsilon_B\} = [D_B]\sum_{m=1}^{q}[B_B^{[m]}]\{d_w^{[m]}\} \tag{29}$$

$$\{\sigma_M\} = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix} = [D_M]\{\epsilon_M\} = [D_M]\sum_{m=1}^{q}[B_M^{[m]}]\{d_{uv}^{[m]}\} \tag{30}$$

With:

$$[D_B] = \begin{bmatrix} D_x & D_1 & 0 \\ D_1 & D_y & 0 \\ 0 & 0 & D_{xy} \end{bmatrix} \tag{31}$$

and

$$[D_M] = \begin{bmatrix} \frac{E_x}{1-v_x v_y} & \frac{v_x E_y}{1-v_x v_y} & 0 \\ \frac{v_x E_y}{1-v_x v_y} & \frac{E_x}{1-v_x v_y} & 0 \\ 0 & 0 & G_{xy} \end{bmatrix} \tag{32}$$

3.5 Elastic Buckling

For a given edge traction the geometric stiffness matrix scales linearly, resulting in the following eigenvalue problem:

$$[K_e][X] = [\Lambda][K_g][X] \tag{33}$$

where [Λ] is a diagonal matrix containing the eigenvalues (buckling loads) and [X] is a fully populated matrix containing the eigenmodes (buckling modes) in its columns. The solution is easily found by multiplying the elastic stiffness matrix with the inverse of the geometric stiffness matrix and calculating the eigenvalues and eigenvectors of the resulting matrix. Fortunately there are an abundance of JAVA libraries available, such as JAMA, that provide this functionality as well as other matrix and vector operations.

For the simply supported case, the solutions for any m are independent. This is due to the orthogonality of K_e and K_g. Further, the buckling load for any m may be found by performing the solution for $m = 1$ at a length of a/m. As a result, it has become a custom to express FSM results in terms of the minimum buckling load at various half-wave lengths, L, as opposed to FEM where a model is typically analysed for many buckling loads at a fixed length.

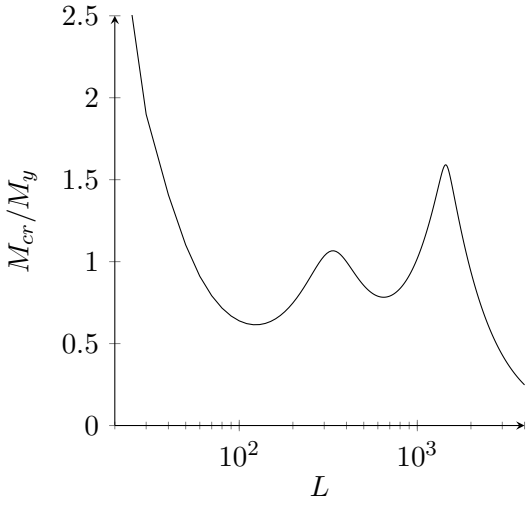


Figure 5: Typical signature curve for C-section.

For all boundary conditions except simply supported, the orthogonality of K_e and K_g is lost. The FSM solution can no longer be expressed in terms of the minimum buckling load at various half-wave lengths but instead should be interpreted as the minimum buckling load for all m's at various physical lengths. In these cases, FSM has the same buckling mode identification problem as FEM.

3.6 Procedure for Modelling Physical Problems Using FSM

The procedure for modelling physical problems using FSM typically follows the same procedure as for FEM, except for one important difference. The longitudinal shapefunctions used in FSM change depending on the boundary conditions of the physical problem, whereas in FEM a single shapefunction is typically used regardless of the boundary conditions. The first step towards finding a solution would be to choose or calculate a longitudinal shapefunction that represents the boundary conditions of the pysical problem accurately. The available shapefunctions have been explicitly calculated in literature and are listed in Section 3.1.1. For example, if the physical problem's boundary conditions are assumed as simply-supported, the longitudinal shapefunction would be as follows:

$$Y_m(y) = \sin \frac{\mu_m y}{a} \qquad (\mu_m = \pi, 2\pi, 3\pi, ...m\pi) \qquad \text{(4 revisited)}$$

Where $m = 1, 2, ..., q$

With closer inspection of Equation 4 one realizes that there are two unknowns, a and q. q represents the maximum number of longitudinal terms used in the series function. Choosing q as a larger number improves the accuracy of the result because the series function more accurately represents the shape of the solution. Taking $q = 10$ usually delivers reasonably accurate results. The unknown a represents the length of the physical model, or in other words the distance between the end conditions.

The second step would be to determine the material parameters of the member under consideration. These are: the Young's Modulus, E, shear modulus, G, and Poisson's ratio, v.

Thirdly, the cross-section of the physical problem is discretized into a set of node and a set of strips, so that each strip has a known width, b, and thickness t and each node has known coordinates.

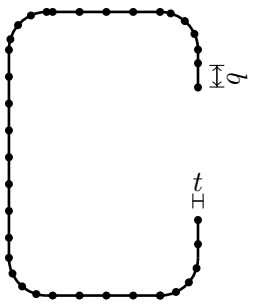


Figure 6: Discretized channel section profile.

The fourth step would be to determine the load vectors and stifness matrices for all strips. Since the series function, Y, length, a, and width b for each strip is now known, we only need the size of the distributed load in the x and y direction acting on each strip. Then the equations from Section 3.3 can be employed to calculate the load vectors. After the material parameters, width, thickness, length

and series function is known, the stiffness matrix for each strip may be calculated as described in Section 3.2.

The individual stiffness matrix of a strip is computed from its material properties and dimensions. This is done with respect to its local coordinate system. For these strips to act as a single structure they need to be assembled. To do this each individual stiffness matrix must be first transformed to the global coordinate system as follows:

$$[k] = [R]^T [k'][R] \tag{34}$$

in which $[R]$ is the transformation matrix

$$[R] = \begin{bmatrix} [r] & \cdot \\ \cdot & [r] \end{bmatrix} \tag{35}$$

with

$$[r] = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{36}$$

where β is the angle between the local and global coordinate systems taken as clockwise positive. The directional cosines can be calculated directly from the nodal coordinates of a strip as follows:

$$\cos\beta = \frac{x_2 - x_1}{\sqrt{(x_2 - x_1)^2 + (z_2 - z_1)^2}} \tag{37}$$

and

$$\sin\beta = \frac{z_2 - z_1}{\sqrt{(x_2 - x_1)^2 + (z_2 - z_1)^2}} \tag{38}$$

where x_i the x-coordinate and z_i the z-coordinate of node i respectively.

Assembling the finite strip stiffness matrix is more complex than in the finite element method. For simply supported strips, the terms of the series are uncoupled and the stiffness matrices for each term can be formed, assembled and solved separately. In this case the same process as in FEM is followed. Thus if nodes 1 and 2 of strip (i) are associated with nodes I and J of the structure respectively, then for the mth term of the series, the four sub-matrices of the stiffness matrix $\left[k^{[mm]}\right]_{(i)}$ will be added into the framework of the overall stiffness matrix as follows:

where:

$$\left[k^{[mm]}\right]_{(i)} = \begin{bmatrix} [S_{11}]_{mm(i)} & [S_{12}]_{mm(i)} \\ [S_{21}]_{mm(i)} & [S_{22}]_{mm(i)} \end{bmatrix} \tag{39}$$

Table 2: Assembly of S-S strip into system stiffness matrix.

Nodes	I-1	I	I+1 ... J-1	J	J+1
I-1					
I		$[S_{11}]_{mm(i)}$		$[S_{12}]_{mm(i)}$	
I+1 ... J-1					
J		$[S_{21}]_{mm(i)}$		$[S_{22}]_{mm(i)}$	
J+1					

Assembly of the system load vector procedes in a similar manner. Thus for the same strip (i) with nodes 1 and 2 corresponding to nodes I and J of the structure respectively, then for the mth term of the series, the two sub-vectors of the load vector, $\{F^{[m]}\}_{(i)}$ will be added into the framework of the system load vector as follows:

Table 3: Assembly of S-S strip's load vector into system load vector.

Nodes	
I-1	
I	$\{F_1\}_{m(i)}$
I+1 ... J-1	
J	$\{F_2\}_{m(i)}$
J+1	

where:

$$\{F^{[m]}\}_{(i)} = \begin{Bmatrix} \{F_1\}_{m(i)} \\ \{F_2\}_{m(i)} \end{Bmatrix} \tag{40}$$

For example consider the simply-supported system as shown in Figure 8. This system consists of three nodes, nodes 1, 2 and 3 and two strips, strip 1 and strip 2. Strip 1 has two nodes, node 1 and node 2 and strip 2 has two nodes, node 2 and node 3.

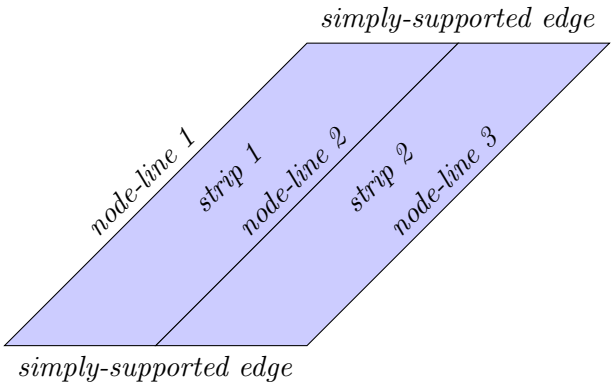


Figure 7: S-S strip assembly.

Assembly of the system stiffness matrix and load vector for term m will commence in a similar manner to FEM as follows:

Table 4: Assembly of S-S strip into system stiffness matrix.

Nodes	1	2	3
1	$[S_{11}]_{mm(1)}$	$[S_{12}]_{mm(1)}$	$[0]$
2	$[S_{21}]_{mm(1)}$	$[S_{22}]_{mm(1)} + [S_{11}]_{mm(2)}$	$[S_{12}]_{mm(2)}$
3	$[0]$	$[S_{21}]_{mm(2)}$	$[S_{22}]_{mm(2)}$

Table 5: Assembly of S-S strip's load vector into system load vector.

Nodes	
1	$\{F_1\}_{m(1)}$
2	$\{F_2\}_{m(1)} + \{F_1\}_{m(2)}$
3	$\{F_2\}_{m(2)}$

Hence, for each $m = 1, .., q$ the system equation that needs to be solved becomes:

$$\begin{bmatrix} [S_{11}]_{mm(1)} & [S_{12}]_{mm(1)} & [0] \\ [S_{21}]_{mm(1)} & [S_{22}]_{mm(1)} + [S_{11}]_{mm(2)} & [S_{12}]_{mm(2)} \\ [0] & [S_{21}]_{mm(2)} & [S_{22}]_{mm(2)} \end{bmatrix} \begin{Bmatrix} \{d^{[m]}\}_{node1} \\ \{d^{[m]}\}_{node2} \\ \{d^{[m]}\}_{node3} \end{Bmatrix} = \begin{Bmatrix} \{F_1\}_{m(1)} \\ \{F_2\}_{m(1)} + \{F_1\}_{m(2)} \\ \{F_2\}_{m(2)} \end{Bmatrix} \quad (41)$$

The above may then be solved by use of any appropriate solution method. Recalling that the value of q has been chosen as to provide a solution with reasonable accuracy, convergence is usually achieved with a value of between 5 to 10. For each m term the solution is stored and added to the a nodal parameter vector $\{d\}$.

For strips with end conditions other than simply supported, terms in the series function $Y(y)$ will be coupled. The stiffness matrices for each term can no longer be formed, assembled and solved separately. The stiffness matrices for the various terms must be assembled into a single stiffness matrix. Thus if nodes 1 and 2 of strip (i) is associated with nodes I and J of the structure respectively, then for the mth and nth terms of the series, the four sub-matrices of the stiffness matrix $\left[k^{[mn]}\right]_{(i)}$ will be added into the framework of the overall stiffness matrix as follows:

Table 6: Assembly of general strip into system stiffness matrix.

Nodes		I-1	I			I+1 ... J-1	J			J+1
	Terms		1...	n	...q		1...	n	...q	
I-1										
I	1...									
	m			$[S_{11}]_{mn(i)}$				$[S_{12}]_{mn(i)}$		
	...q									
I+1 ... J-1										
J	1...									
	m			$[S_{21}]_{mn(i)}$				$[S_{22}]_{mn(i)}$		
	...q									
J+1										

where:

$$\left[k^{[mn]}\right]_{(i)} = \begin{bmatrix} [S_{11}]_{mn(i)} & [S_{12}]_{mn(i)} \\ [S_{21}]_{mn(i)} & [S_{22}]_{mn(i)} \end{bmatrix} \tag{42}$$

Assembly of the system load vector procedes in a similar manner. Thus for the same strip (i) with nodes 1 and 2 corresponding to nodes I and J of the structure respectively, then for the mth term of the series, the two sub-vectors of the load vector, $\{F^{[m]}\}_{(i)}$ will be added into the framework of the system load vector as follows:

Table 7: Assembly of general strip's load vector into system load vector.

Nodes	Terms	
I-1		
I	1...	
	m	$\{F_1\}_{m(i)}$
	...q	
I+1 ... J-1		
J	1...	
	m	$\{F_2\}_{m(i)}$
	...q	
J+1		

where:

$$\{F^{[m]}\}_{(i)} = \begin{Bmatrix} \{F_1\}_{m(i)} \\ \{F_2\}_{m(i)} \end{Bmatrix} \tag{43}$$

For example consider the system as shown in Figure 8. This system is clamped at both ends, consist of three nodes, nodes 1,2 and 3 and two strips, strip 1 and strip 2. Strip 1 has two nodes, node 1 and node 2 and strip 2 has two nodes, node 2 and node 3. For this example it is assumed that a reasonable accuracy will be achieved with only two longitudinal terms i.e. $q = 2$.

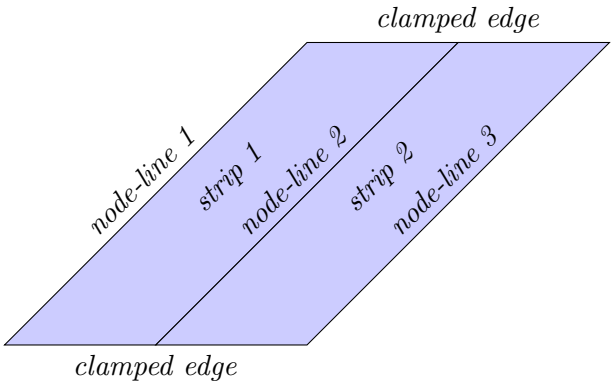


Figure 8: Fixed strip assembly.

Assembly of the system stiffness matrix and load vector for $m = 1, 2$ will commence as follows:

Table 8: Assembly of C-C strip into system stiffness matrix.

Nodes		1		2		3	
	Terms	1	2	1	2	1	2
1	1	$[S_{11}]_{11(1)}$	$[S_{11}]_{12(1)}$	$[S_{12}]_{11(1)}$	$[S_{12}]_{12(1)}$		
	2	$[S_{11}]_{21(1)}$	$[S_{11}]_{22(1)}$	$[S_{12}]_{21(1)}$	$[S_{12}]_{22(1)}$		
2	1	$[S_{21}]_{11(1)}$	$[S_{21}]_{12(1)}$	$[S_{22}]_{11(1)} + [S_{11}]_{11(2)}$	$[S_{22}]_{12(1)} + [S_{11}]_{12(2)}$	$[S_{12}]_{11(2)}$	$[S_{12}]_{12(2)}$
	2	$[S_{21}]_{21(1)}$	$[S_{21}]_{22(1)}$	$[S_{22}]_{21(1)} + [S_{11}]_{21(2)}$	$[S_{22}]_{22(1)} + [S_{11}]_{22(2)}$	$[S_{12}]_{21(2)}$	$[S_{12}]_{22(2)}$
3	1			$[S_{21}]_{11(2)}$	$[S_{21}]_{12(2)}$	$[S_{22}]_{11(2)}$	$[S_{22}]_{12(2)}$
	2			$[S_{21}]_{21(2)}$	$[S_{21}]_{22(2)}$	$[S_{22}]_{21(2)}$	$[S_{22}]_{22(2)}$

Table 9: Assembly of C-C strip's load vector into system load vector.

Nodes	Terms	
1	1	$\{F_1\}_{1(1)}$
	2	$\{F_1\}_{2(1)}$
2	1	$\{F_2\}_{1(1)} + \{F_1\}_{1(2)}$
	2	$\{F_2\}_{2(1)} + \{F_1\}_{2(2)}$
3	1	$\{F_2\}_{1(2)}$
	2	$\{F_2\}_{2(2)}$

Unfortunately the system equation becomes to large to be shown here. The equation may be solved by use of any appropriate solution method. From the above it should be clear why the FSM is primarily used for the solution of simply-supported structures. The stiffness matrices quickly become large with the introduction of more longitudinal terms, in turn reducing the computational efficiency of the solution. For elastic buckling problems, one would follow the same assembly procedure but, for both the geometric and elastic stiffness matrices and only for a maximium number of longitudinal terms $q = 1$. The load vector is not applicable for these problems since the load is in the form of a compressive edge traction applied at the boundary, and is incorporated in the geometric stiffness matrix. The system equation then boils down to finding the eigenvectors and eigenvalues of $KeKg^{-1}$ repeatedly while varying the length of the model, a.

An interesting fact about FSM is that the system equations can be solved without explicit introduction of any boundary conditions since the boundary conditions are satisfied *a priori* by the series function, Y. Those familiar with FEM will know that when an attempt is made to solve a system without specifying boundary conditions, the stiffness matrix becomes non-positive definite. However, this is not the case with FSM.

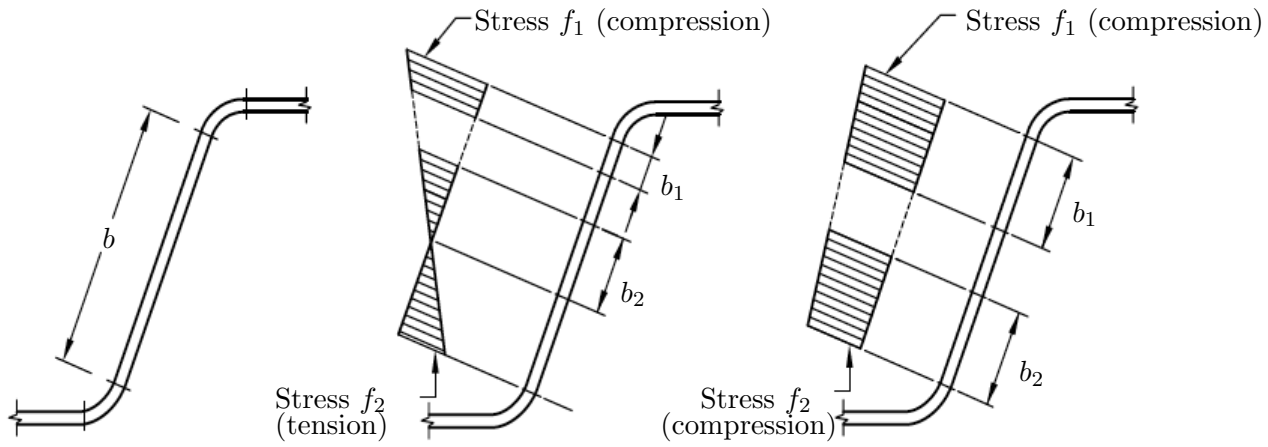
4 Design Methods

The challenge for any design method is accounting for all the complicated phenomena arising when using thin walled cold-formed sections in compression while still being as simple as possible. In the current specification, SANS 10162-2 [6], two methods for the design of cold-formed members exist the Effective Width Method and the Direct Strength method.

4.1 The Effective Width Method (EWM)

The Effective Width Method (EWM) is a well-known and trusted design procedure. Its basis is well explained in textbooks and specifications [13] [14] [15]. The basic idea is that if the plates comprising a cross-section are subjected to local buckling, their effectiveness is decreased. This loss of effectiveness is accounted for by reducing the width of the plates subjected to buckling.

A slender plate under compressive load is able to support loads greater than that which causes it to buckle. This additional load is carried by the plate after buckling by means of transverse stresses. This phenomenon is called "post-buckling reserve" [2]. When this happens, the stress distribution in the plate becomes non-linear. To simplify the problem, the EWM assumes a linear stress distribution on an effective plate rather than the actual plate with the actual non-linear longitudinal stress distribution that develops due to buckling. In this way, every plate or element comprising a cross section is reduced to its effective width, and so the strength of the entire cross-section is reduced.



(a) Actual element (b) Effective element and stress on effective element

Figure 9: Effective width of a plate in compression. Adapted from SANS 10162-2 [6]

4.2 The Direct Strength Method (DSM)

The DSM simplifies the design approach by relying on signature curves, see figure 10), instead of effective width. These strength curves are obtained by performing an elastic buckling analysis using the Finite Strip Method (FSM). The buckling loads are obtained based on the entire member cross-section rather than individual plates. As a result, DSM does not ignore element interaction like the Effective Width Method does.

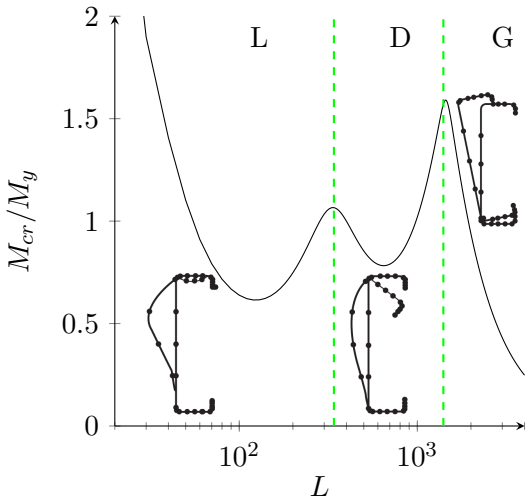


Figure 10: Signature curve for C-section.

Thin-walled members typically have three buckling modes, shown in figure 10, of interest in design: local (L), distortional (D) and global (G) buckling. SANS 10162-2 [6] defines the three buckling modes as follows:

- Local buckling - A mode of buckling involving plate flexure alone without transverse deformation of the line or lines of intersection of adjoining plates.
- Distortional buckling - A mode of buckling involving a change in cross-sectional shape, excluding local buckling.
- Global / Flexural-torsional buckling - A mode of buckling in which compression members can bend and twist simultaneously without change of cross-sectional shape.

The DSM has two main requirements. Firstly, accurate modelling of member stability is crucial to the success of a Direct Strength Method design. Secondly, the local minima of the buckling load versus length curve corresponding to local, distortional and global buckling should be identifiable as in figure 10. This is done by separation of the modes and unless specialized techniques are used, the separation can not be performed by conventional Finite Element analysis.

There is ongoing research to extend this method further to include members subjected to elevated temperatures [16] (e.g. during fire conditions) and members with holes [10]. However, for basic cold-

formed member design, it is a sufficient design procedure that requires less effort and is a better predictor of actual member strength than the effective width method.

It is to be noted that the cross-sections modelled with FSM typically consist of straight lines. In reality, most cold formed sections have curved edges, so naturally, there will exist a difference in the cross-section properties of the modelled cross-section versus the real cross-section. Specifically, the values for the moment of inertia Ixx and cross-sectional area A will differ. One must, therefore, be careful when using FSM software that calculates these cross-section properties from the straight line model, as this could give a false impression of the member strength.

Consider the following cold-formed lipped channel with section designation 75×50×20×3, and its straight line model counterpart.

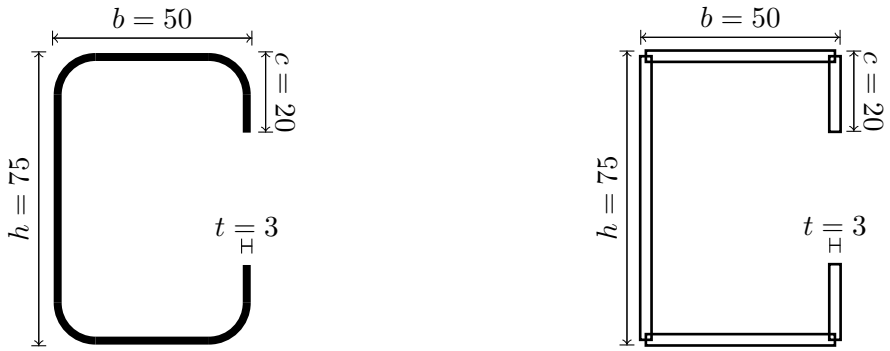


Figure 11: Channel section and its straight line model counterpart.

The section properties of the channel as given in South African Steel Construction Handbook [21] are compared to section properties calculated for the straight line model.

Table 10: Actual versus modelled cross-section properties

Property	Actual	Modelled	% difference
Area	578 mm^2	609 mm^2	5.36
Moment of inertia (strong axis)	0.500×10^6 mm^4	0.542×10^6 mm^4	8.40
Moment of inertia (weak axis)	0.204×10^6 mm^4	0.223×10^6 mm^4	8.40
Section modulus (strong axis)	13.3×10^3 mm^3	14.4×10^3 mm^3	9.31
Section modulus (weak axis)	6.98×10^3 mm^3	7.68×10^3 mm^3	10.03

From Table 10 it is clear that there is a large difference between the actual and modelled channel sections' cross-sectional properties. The significance of these differences can be seen when the capacities of these sections are calculated at yield. For example, assume that the yield stress of the material is $F_y = 355$ MPa. The axial load and moment that would cause each cross-section to yield are calculated from the section properties and compared in Table 11.

Table 11: Actual versus modelled strength

Load	Actual	Modelled	% difference
Yield moment (strong axis)	4.72 kN.m	5.11 kN.m	8.26
Yield moment (weak axis)	2.48 kN.m	2.73 kN.m	10.08
Yield axial load	205 kN	216 kN	5.37

From Table 11 it is apparent that the straight line model is significantly stronger than the actual member. This states the importance of creating a model that represents the actual member accurately. When high levels of accuracy are not necessary one must at least use the actual member's cross-section properties instead of the modelled member for calculating yield loads as inputs into DSM.

DSM is based on the same empirical assumptions as the Effective Width Method, that the nominal strength is a function of the elastic buckling load and the yield strength of the material. The DSM equations were calibrated against a large amount of experimental data, similar to the Effective Width Method. In fact, many of the same experiments were employed. [2]

When the Direct Strength Method was developed, it was decided that users of the method should be aware of the cross-sections employed to verify the approach. Thus the idea of pre-qualified sections was established. The implication being that members falling within the geometrical bounds of the pre-qualified set may be designed with partial factors, ϕ, whereas unqualified members are designed using slightly more conservative factors.

Members with perforations/holes i.e. that do not have a uniform cross section cannot be modelled by conventional FSM, although there is ongoing development to extend the DSM to such members [10]. Landesmann and Camotim [16] have shown that it is possible to predict ultimate strength of columns/studs in fire condition using the Direct Strength Method.

4.3 Direct Strength Method Formulae

The relevant DSM formulae in the specification [6] Section 7 are listed below.

4.3.1 Axial Strength

The nominal axial strength, P_n, is the minimum of the individual predicted capacities:

$$P_n = \min(P_{ne}, P_{nl}, P_{nd}) \tag{44}$$

Where:

P_{ne} = The nominal axial strength for global buckling.

$$P_{ne} = \begin{cases} (0.658^{{\lambda_c}^2})P_y & \text{if } \lambda_c \leq 1.5, \\ \dfrac{0.877}{{\lambda_c}^2}P_y & \text{if } \lambda_c < 1.5 \end{cases} \tag{45}$$

P_{nl} = The nominal axial strength for local buckling.

$$P_{nl} = \begin{cases} \left[1 - 0.15(P_{crl}/P_{ne})^{0.4}\right]\left(\dfrac{P_{crl}}{P_{ne}}\right)^{0.4} P_{ne} & \text{if } \lambda_l > 0.776, \\ P_{ne} & \text{if } \lambda_l \leq 0.776 \end{cases} \tag{46}$$

P_{nd} = The nominal axial strength for distortional buckling.

$$P_{nd} = \begin{cases} \left[1 - 0.25(P_{crd}/P_y)^{0.6}\right]\left(\dfrac{P_{crd}}{P_y}\right)^{0.6} P_y & \text{if } \lambda_d > 0.561, \\ P_y & \text{if } \lambda_d \leq 0.561 \end{cases} \tag{47}$$

Where, $\lambda_c = \sqrt{P_y/P_{cre}}$, $\lambda_l = \sqrt{P_{ne}/P_{crl}}$, $\lambda_d = \sqrt{P_y/P_{crd}}$, $P_y = A_g F_y$, P_{cre}, minimum of the critical elastic column buckling load in flexural, torsional, or torsional-flexural buckling, P_{crl}, critical elastic local column buckling load, P_{crd}, critical elastic distortional column buckling load, A_g, the gross area of the cross-section, and F_y, the yield stress.

4.3.2 Bending Strength

The nominal bending strength, M_n, is the minimum of the individual predicted capacities:

$$M_n = \min(M_{ne}, M_{nl}, M_{nd}) \tag{48}$$

Where:

M_{ne} = The nominal bending strength for global buckling.

$$M_{ne} = \begin{cases} M_{cre} & \text{if } M_{cre} < 0.56M_y, \\ \dfrac{10}{9}M_y\left(1 - \dfrac{10M_y}{36M_{cre}}\right) & \text{if } 2.78My \geq M_{cre} \geq 0.56M_y \\ M_y & \text{if } M_{cre} > 2.78M_y \end{cases} \tag{49}$$

M_{nl} = The nominal bending strength for local buckling.

$$M_{nl} = \begin{cases} \left[1 - 0.15(M_{crl}/M_{ne})^{0.4}\right]\left(\dfrac{M_{crl}}{M_{ne}}\right)^{0.4} M_{ne} & \text{if } \lambda_l > 0.776, \\ M_{ne} & \text{if } \lambda_l \leq 0.776 \end{cases} \tag{50}$$

M_{nd} = The nominal bending strength for distortional buckling.

$$M_{nd} = \begin{cases} \left[1 - 0.22(M_{crd}/M_y)^{0.5}\right]\left(\dfrac{M_{crd}}{M_y}\right)^{0.5} M_y & \text{if } \lambda_d > 0.673, \\ M_y & \text{if } \lambda_d \leq 0.673 \end{cases} \tag{51}$$

Where, $\lambda_l = \sqrt{M_{ne}/M_{crl}}$, $\lambda_d = \sqrt{M_y/M_{crd}}$, $M_y = Z_f F_y$, M_{cre}, critical elastic column buckling moment, M_{crl}, critical elastic local column buckling moment, M_{crd}, critical elastic distortional column buckling moment, Z_f, the gross section modulus referenced to the extreme fiber in the first yield, and F_y, the yield stress.

5 A comparison of the DSM and EWM

In this section the DSM and EWM wil be compared by looking at three aspects namely: design effort, accuracy and economy.

5.1 Design effort

The EWM calculations as given by SANS 10162-2 [6] consist of a set of closed-form expressions. The elements comprising a cross-section are categorised into either unstiffened or stiffened elements. Stiffened elements are further categorised by their means of stiffening. A stiffened element may be edge-stiffened, have single a intermediate stiffener, or have multiple intermediate stiffeners. SANS 10162-2 [6] also makes a distinction between flat elements, bends and arched elements. The designer is required to identify each of these types of elements as well as the cross section type of the overall member and apply the formulae from the relevant section of the specification. In this way the effective width for every element in the cross-section is determined. The calculations may range from quick and simple to involved and time-consuming depending on the complexity of the member cross-section. The exercise usually includes the calculation of several section properties. An implication being that the section properties, if not found in some table, need to be calculated by hand or with software. Specifically, the design of more optimized cross-sections e.g. by the introduction of longitudinal plate-stiffeners may get quite involved and time-consuming if the design is performed by means of the EWM, because of the large amount of elements that these types of cross-sections consist of.

The equations utilized in the Direct Strength Method are far simpler than those employed in the Effective Width Method. Moreover, the equations and amount of work stay the same regardless of the type of cross-section that is designed for. If any cross-section properties are to be calculated, the modelling software usually includes some tool that does so. Because the hand calculations are typically much less than when using the Effective Width Method, human error is eliminated to a certain extent.

The Direct Strength Method Design Guide [3] provides numerous examples demonstrating the use of DSM for a range of cross-sections. The examples it provides are intended to show that when an elastic buckling analysis tool is available, the DSM requires less calculation and complexity than the EWM. The guide notes that for example, the bending strength calculation of a lipped channel section takes 4.5 pages using the EWM while the same calculation performed using DSM takes less than 2 pages.

5.2 Accuracy

The reliability of a design method is the probability that the method will deliver a satisfactory result under various conditions. Freitas, Brandão and Freitas [9] calculated resistance factors for cold-formed steel columns using a first-order second-moment reliability approach. A test database of 323 cold-formed steel columns was formed, and test-to-predicted statistics were obtained for the Effective Width, Effective Section, and Direct Strength methods. Freitas et al. found that while the calculated resistance factors for all three methods were consistent with the value specified by the Brazilian code, DSM outperforms the other two methods in terms of a reliability index β, where β can be understood as a measure of safety.

Table 12: Reliability index comparison

	Reliability Index β			
	$1.2D_n + 1.6L_n$		$1.25D_n + 1.5L_n$	
	$\frac{D_n}{L_n} = 0.2$	$\frac{D_n}{L_n} = 1/3$	$\frac{D_n}{L_n} = 0.2$	$\frac{D_n}{L_n} = 1/3$
EWM	2.58	2.62	2.41	2.47
ESM	2.65	2.70	2.47	2.54
DSM	2.71	2.76	2.53	2.60

Schafer [7] established the reliability of the Direct Strength Method using the limit states design format in use in the United States: Load and Resistance Factor Design (LRFD). A target reliability index β of 2.5 was employed. It was found that the reliability index of the Direct Strength Method is as good, or better than the Effective Width Method.

Schafer [7] compared the strength predictions of the Direct Strength and Effective Width methods as a function of web height over width in a channel section column. He found that the Effective Width solution becomes less conservative as the slenderness of the web increases. This behaviour is exaggerated by a certain property of typically available channel sections, specifically that channel sections with deeper webs have flanges that are approximately the same width as channel sections with more shallow webs. The Effective Width Method treats the flanges and web separately, thus the slenderness of the web does not influence the solution of the flanges. Peköz [5] states that the variation in the test-to-predicted ratio for the Effective Width Method is high. Furthermore, results by researchers indicate consistently unconservative strength predictions for certain classes of channels and Z sections when the EWM is used.

Unlike the Effective Width Method, the Direct Strength Method includes interaction between the web

and the flanges. As a result, the Direct Strength Method performs more reliably over the full spectrum of web-slenderness. This shows that the interaction between plate members is crucial in accurately predicting the strengths of these members.

Schafer [7] states that the Effective Width Method ignores inter-element equilibrium. He continues to explain that this is due to the fact that the Effective Width Method calculation considers every plate or element of the section geometry individually. The non-linear stress distribution, approximated by the linear stress distribution in the Effective Width Method, is an approximation itself. It represents the average of the longitudinal membrane stress and ignores the variation in stress along the thickness and length of the plate. Thus determining the true "effective width" of a plate is more complicated than assumed in the EWM design procedure.

The inclusion of interaction between web and flanges can also work against the Direct Strength Method but only when taken to extremes, Schafer [7] notes. As the slenderness of one part of the member increases, the elastic critical buckling stress of the member will approach zero. Consequently, the strength predicted by the Direct Strength Method will also approach zero. This is a fundamental limitation of the Direct Strength Method and was reported in the first paper to propose the approach [5]. The limitation causes the Direct Strength Method to be overly conservative in predicting strengths when the cross section of a member includes a very slender element. In such a situation Effective Width Method will only assume the slender element to have zero strength instead of the whole cross section. Hat or deck sections in bending usually fall within this category because of their low yield stress and slender compression flanges without intermediate stiffeners. The Direct Strength Method provides overly conservative results for these sections while the Effective Width Method delivers more reasonable predictions [7]. However, ignoring the interaction between elements is generally not a good idea.

5.3 Economy

By comparing the factored strength predicted by the EWM and DSM, we can determine which method delivers a more economical section. From the previous discussion, it can be expected that for C-sections with larger web dimensions the EWM will tend to deliver more economical results as the solution becomes less conservative.

Table 13: Strength comparison

	EWM		DSM	
	ϕMn kN.m	ϕPn kN	ϕMn kN.m	ϕPn kN
Lipped C				
Major-axis bending (braced)	10.62		9.49	
Compression (braced)		92.08		73.40
Hat section				
Minor-axis bending	7.68		8.02	
Compression (braced)		329.17		309.15

Where: ϕMn = Factored nominal bending capacity.
ϕPn = Factored nominal capacity under axial load.

From Table 13 it is clear that the EWM delivers a 10.64% more economical (although less conservative) C- section than the DSM in bending and is 20.29% more economical in axial compression. Although, for a hat section in bending the DSM would deliver a 4.24% more economical member but 6.08% less economical in axial compression.

5.3.1 Discussion

The Effective Width method has been adopted in many specifications worldwide. However, it has two main drawbacks. Both arise from the fact that the Effective Width Method treats elements/-plates of the whole cross-section independently. One, interaction between elements/plates is ignored. Two, when sections get more optimized (e.g., through the introduction of longitudinal stiffeners) more plates/elements are introduced resulting in an overly complicated and time-consuming calculation.

Although the Effective Width Method is a useful design model, the calculations it entails may become cumbersome. This could cause engineers to steer away from the use of a certain material. Schafer [7]

states that the Effective Width Method (i) ignores inter-element equilibrium, (ii) incorporation of competing buckling modes can be awkward and (iii) determining the effective section becomes overly complicated as more optimized sections are used.

The Direct Strength Method simplifies the design approach by relying on strength curves instead of effective width. Elastic buckling loads are obtained based on the entire member cross-section rather than individual plates. According to DSM, if all the buckling loads or moments (i.e., Local, Distortional and Global) are known, as well as the yield load, the strength of the member can be predicted at each buckling mode. The nominal strength of the member is then taken as the minimum of the predicted strength at the three buckling modes. Because DSM considers the entire cross-section rather than individual plates, the calculations it requires are faster and simpler than the Effective Width calculations. However, DSM does require special tools, for example, a Finite Strip Model, that could be intimidating to implement. Fortunately, a few tools already exist so designers do not need to implement FSM themselves. This also makes the Direct Strength Method less prone to human error because fewer calculations are done by hand.

With technology becoming better every day there are continuously faster and better ways for engineers to solve problems. Engineers should continuously be adapting and learning new methods. With the above comparison of the Effective Width Method and the Direct Strength Method in mind, it is postulated that the Direct Strength Method yields designs that are more reliable than the EWM. Although the method is still lacking in some areas, there is a plethora of research being done to make the method better and more universal.

6 FSM Implementation

Every Finite Strip model has basic requirements for the functionality it should provide. For instance, all Finite Strip models consist of a set of nodes and strips that define the model's geometry. Therefore it makes sense to create an object model consisting of these basic components such as nodes and strips and the model itself wherein these components are contained.

The following section describes how the Finite Strip problem is mapped from ordinary mathematical terms into an object model that can be used on a computer. Note that the JAVA types Vector and Matrix and any methods that perform matrix operations used in this source code are defined in a package developed by Stellenbosch University and are not native to JAVA.

6.1 JAVA Object Definitions

6.1.1 Node

Nodes are the most basic components of a modelled member. A Node is unique and should have a unique identifier of type int. To ensure the identifiers are unique, a static int is employed that is incremented each time a Node is created. The unique identifier then gets its value from the static variable. Together, a set of nodes defines the cross-section geometry of the member. A Node represents a single node line in the FSM theory, it is also a point on a member's two-dimensional cross-section and as such should have a two-dimensional coordinate. There is a total of four degrees of freedom that may be prescribed at a node line. A Node will account for this by having a boolean array of size 4, each boolean representing whether the corresponding degree of freedom is free (false) or prescribed (true). For each series term m, a different set of nodal parameters i.e. u, v, w and θ are computed at each node. These parameters will be stored as a set of vectors, each mapped to its corresponding term number.

The constructor of a Node is as follows:

```
1 public Node(double xCoord ,double zCoord, Model model)
```

where: xCoord and zCoord are the x and z coordinates of the node and model is the Model to which this node belongs. After a Node has been created the following methods may be used to change or retrieve the node's location as well as retrieve its ID:

```
1 public void setXCoord(Double xCoord) // set the x-coordinate of the node
2 public void setZCoord(Double zCoord) // set the z-coordinate of the node
3 public double getXCoord() // return the x-coordinate of the node
4 public double getZCoord() // return the z-coordinate of the node
5 public int getNodeId() // return the node ID
```

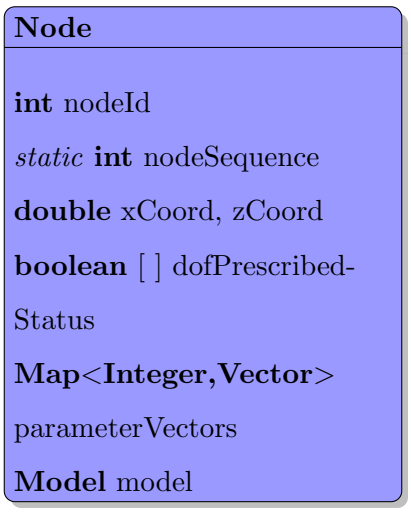


Figure 12: Object diagram for a Node object.

The degrees of freedom at the node line may be retrieved or set by calling:

```
public boolean [] getStatus() // return status array
public void setStatus(boolean [] status) // set degree of freedom statusses (u, v, w,
    theta)
```

The status array is a one-dimensional array of length 4 and should be interpreted as being in the sequence $[u\ v\ w\ \theta]$. For instance, if rotation alone is to be permitted at the node line, then the status array would read [true true true false]. The nodal displacements corresponding to each m-term are stored in the Nodes. These are stored and can be retrieved by use of the following two methods:

```
public void setParameterVector(Vector P, int m )  // set the parameter vector for the
    given m-term at this node
public Vector getParameterVector(int m)  // return the parameter vector for given m-
    term
```

Where each parameter vector is in the form:

$$\begin{Bmatrix} u_{[m]} \\ v_{[m]} \\ w_{[m]} \\ \theta_{[m]} \end{Bmatrix}$$

and m such vectors exist.

6.1.2 Material

A Material object represents the material of a modelled member. It contains important characteristics such as the material's name, Young's modulus, Poisson ratio, shear modulus, and yield stress.

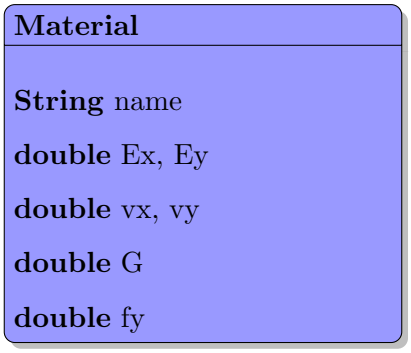


Figure 13: Object diagram for a Material object.

The Material class contains the necessary get and set methods to retrieve or change each of its attributes.

6.1.3 Series

A strip may have different end conditions as dictated by the series function employed. Different series functions all share some common attributes such as domain length, a, which is equal to the length of the model. Therefore it makes sense to have an abstract class that contains attributes common to all series functions and dictates how the functions should be used. Series is an abstract class. It contains an attribute for domain length and another that indicates whether the series represents simple boundary conditions and therefore can be handled as a special case of the FSM problem.

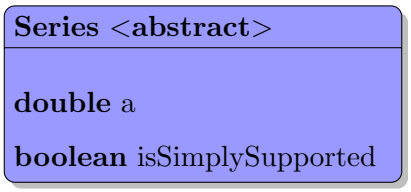


Figure 14: Object diagram for a Series object.

Subclasses of class Series are forced to complete the following abstract methods:

```
1 public abstract double getFunctionValue(double y,int m);
```

All series functions contain the independent variable y and a term number m. The getFunctionValue() method should take y and m as input parameters and return the value of the function accordingly. As an example, recall the series function used for simply supported end conditions, Equation 4:

$$Y_m(y) = \sin\frac{\mu_m y}{a} \qquad (\mu_m = \pi, 2\pi, 3\pi, ...m\pi) \qquad \text{(4 revisited)}$$

For a subclass of Series representing Equation 4, the getFunctionValue() method would be completed as follows:

```
public double getFunctionValue(double y, int m) {

        return Math.sin(m * Math.PI * y / a);
    }
```

```
public abstract double getMu_m(int m);
```

All series functions have a parameter μ_m that is unique for each series function and can be described as a function of the term number m. the getMu_m() method takes m as an input parameter and should return the value of μ_m. For a subclass of Series representing Equation 4, the getMu_m() method would be completed as follows:

```
public double getMu_m(int m) {

        return m * Math.PI;
    }
```

```
public abstract double getFirstDerivativeValue(double y, int m);
```

The first derivative of the series function is required to compute membrane strain values. For a subclass of Series representing Equation 4, the getFirstDerivativeValue() method would be completed as follows:

```
public double getFirstDerivativeValue(double y, int m) {
        return Math.cos(m * Math.PI * y / a) * m * Math.PI / a;
    }
```

```
public abstract double getSecondDerivativeValue(double y, int m);
```

The second derivative of the series function is required to compute the bending strain values. For a subclass of Series representing Equation 4, the getSecondDerivativeValue() method would be completed as follows:

```
public double getSecondDerivativeValue(double y, int m) {
        return -Math.sin(m * Math.PI * y / a) * (m * Math.PI / a) * (m * Math.PI/
    a);
    }
```

```
public abstract double [] getIntegralValues(int m, int n);
```

Recall the integrals I1 to I5 that form part of Equation 16:

$$I_1 = \int_0^a Y_m Y_n \, dy,\ I_2 = \int_0^a Y_m'' Y_n \, dy,\ I_3 = \int_0^a Y_m Y_n'' \, dy,\ I_4 = \int_0^a Y_m'' Y_n'' \, dy,\ I_5 = \int_0^a Y_m' Y_n' \, dy,$$

These integrals should be computed by the subclasses of Series and should be returned as an array of size 5. Generally the integrals cannot be calculated analytically and must be dealt with numerically, but for the simply supported case they can easily be expressed analytically and computed as follows:

```
public double[] getIntegralValues(int m, int n) {

        double[] I = new double[5];
        double pi = Math.PI;
        double pi2 = pi * pi;
        double pi4 = pi * pi * pi * pi;

        double m2 = m * m;
        double m4 = m * m * m * m;

        double n2 = n * n;

        if (m == n) {
            I[0] = a / 2;
            I[1] = -m2 * pi2 / a / 2.0;
            I[2] = -n2 * pi2 / a / 2.0;
            I[3] = pi4 * m4 / 2.0 / (a * a * a);
            I[4] = pi2 * m2 / 2.0 / a;
        } else {
            I[0] = 0;
            I[1] = 0;
            I[2] = 0;
            I[3] = 0;
            I[4] = 0;

        }

        return I;
    }
```

Note that in the code snippet above the integrals follow the numbering convention used by Cheung [1] and not Li [2] as used in the equations.

```
6 public abstract double getYmIntegral(int m);
```

and

```
public abstract double getFirstDerivativeIntegral(int m);
```

To calculate the load vector we need the integral of the series function as well as the integral of its derivative. Recall Equation 24:

$$\{F_M^{[m]}\} = \frac{b}{2}\begin{Bmatrix} q_x \int_0^a Y_m \, dy \\ q_y \frac{a}{\mu_y} \int_0^a Y'_m \, dy \\ q_x \int_0^a Y_m \, dy \\ q_y \frac{a}{\mu_y} \int_0^a Y'_m \, dy \end{Bmatrix} \qquad \text{(24 revisited)}$$

The calculation of integrals $\int_0^a Y_m \, dy$ and $\int_0^a Y'_m \, dy$ need to be implemented in subclasses of class Series. For the simply supported series function the methods would be completed as follows:

```
public double getYmIntegral(int m) {
        double pi = Math.PI;
        return a / (pi * m) - (a * cos(pi * m)) / (pi * m);

    }
```

and

```
public double getFirstDerivativeIntegral(int m) {
        return Math.sin(Math.PI * m);
    }
```

6.1.4 BucklingDataPoint

For a given length, a member has a single load at which it buckles. We want to evaluate multiple lengths and get multiple buckling loads for each, so we need a way to store all this data. Multiple eigenvalues (buckling loads) and multiple eigenvectors (buckled shapes) are found from a single Finite Strip solution, but only the minimum load is chosen as well as the vector that corresponds to that load. The BucklingDataPoint object provides a way manage the data.

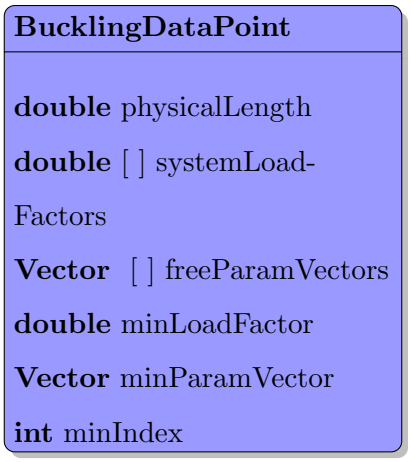


Figure 15: Object diagram for a BucklingDataPoint object.

After a buckling problem has been solved for a given length and m-term, the minimum load factor can be saved in the BucklingDataPoint object by use of the method:

```
1 public void setSystemLoadFactor(int m, double load)
```

The corresponding eigenvector can also be saved by use of the method:

```
1 public void setFreeParamVector(int m, Vector eigVec)
```

The minimum of these values should then be calculated by calling:

```
1 public void calcMinParamAndLoad()
```

If the user wishes to calculate a buckling curve with more than one m-term present, for instance, if they do not want to do a DSM design but rather want to investigate some other plate behaviour, the above method will determine which of the loads are the smallest and also find the buckled shape corresponding to that load.

6.1.5 Strip

The Strip class will be the superclass for all Strip objects. A superclass is needed to accommodate the fact that different types of strips may exist. Note that here, strip (lowercase s) specifically refers to the two-node plate finite strip presented by Cheung while Strip (uppercase S) refers to the JAVA object. In this case, a special Strip-SS will be used for the simply supported end condition and Strip-General for all other boundary conditions. Both will be of type Strip and will extend the superclass Strip. The superclass will make provision for the following properties inherent to all strips: beta - the orientation angle of the strip with regard to the global axis, f1 and f2 - the applied edge tractions at the nodes, two nodes, an ID, thickness and distributed loads.

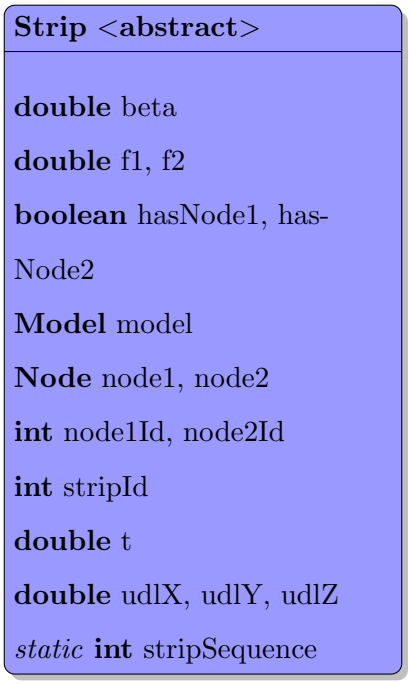


Figure 16: Object diagram for a Strip object.

The abstract class Strip has two constructors. This is to ensure that a Strip may be created either with the Nodes known or unknown.

```
1 public Strip(Node node1, Node node2, Model model)
2 public Strip(Model model)
```

In the case where the nodes are not specified at creation, they need to be added afterward using the methods:

```
1 public void setNode1(Node n)
2 public void setNode2(Node n)
```

The class only has one abstract method:

```
1 public abstract Matrix getStiffnessMatrix(int m, int n);
```

which should be implemented by subclasses of class Strip. This method should return the elastic stiffness matrix in the form given by Equation 14 for any given value of m and n.

The geometric stiffness matrix is calculated by class Strip and is returned in the form given by Equation 18 by use of the following method:

```
public Matrix getGeometricMatrix(int m, int n)
```

The elastic and geometric stiffness matrices returned by the above methods are in strip local coordinates and therefore need to be rotated before assembly by use of Equation 34. Recall Equation 34:

$$[k] = [R]^T[k'][R] \tag{34 revisited}$$

in which $[R]$ is the transformation matrix

$$[R] = \begin{bmatrix} [r] & \cdot \\ \cdot & [r] \end{bmatrix} \tag{35 revisited}$$

with

$$[r] = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{36 revisited}$$

The rotation matrix is calculated by use of the method:

```
public Matrix getRotationMatrix() {

        Matrix R = Matrix.getMatrix(8, 8);
        Matrix r = Matrix.getMatrix(4, 4);

        double s = (node2.getZCoord() - node1.getZCoord()) / getStripWidth(); // sin of
    the strip angle
        double c = (node2.getXCoord() - node1.getXCoord()) / getStripWidth(); // cos of
    the strip angle

        r.clear();

        // the set method is used : set(value , row , column)
        r.set(c, 0, 0);
        r.set(1, 1, 1);
        r.set(c, 2, 2);
        r.set(1, 3, 3);
        r.set(s, 2, 0);
```

```
        r.set(-s, 0, 2);

        int[] ind1 = {0, 1, 2, 3};
        int[] ind2 = {4, 5, 6, 7};

        R.clear();

        R.addSubmatrix(r, ind1);
        R.addSubmatrix(r, ind2);

        return R;

    }
```

and is given as an 8 by 8 matrix in the form of Equation 35.

The displacements at any coordinate in a strip are interpolated from the nodal parameters by use of interpolation functions. Recall the matrices $\left[N_{uv}^{[m]}\right]$ and $\left[N_{w}^{[m]}\right]$ in Equation 10:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \sum_{m=1}^{q} \left[N_{uv}^{[m]}\right] \{d_{uv}^{[m]}\} \qquad \text{and} \qquad \{w\} = \sum_{m=1}^{q} \left[N_{w}^{[m]}\right] \{d_{w}^{[m]}\} \tag{10 revisited}$$

$$\left[N_{uv}^{[m]}\right] = \begin{bmatrix} (1-\bar{x})Y_{[m]} & 0 & (\bar{x})Y_{[m]} & 0 \\ 0 & (1-\bar{x})\dfrac{a}{\mu_{[m]}}Y'_{[m]} & 0 & (\bar{x})\dfrac{a}{\mu_{[m]}}Y'_{[m]} \end{bmatrix} \tag{11 revisited}$$

$$\left[N_{w}^{[m]}\right] = \begin{bmatrix} (1-3\bar{x}^2+2\bar{x}^3) & x(1-2\bar{x}+\bar{x}^2) & (3\bar{x}^2-2\bar{x}^3) & x(\bar{x}^2-\bar{x}) \end{bmatrix} \tag{13 revisited}$$

With $\bar{x} = \dfrac{x}{b}$.

The matrices $\left[N_{uv}^{[m]}\right]$ and $\left[N_{w}^{[m]}\right]$ are calculated by calling the following methods:

```
public Matrix getBendingDisplacementShapeFunctionMatrix(double x, double y, int m) {
        Matrix N = Matrix.getMatrix(1, 4);
        Series Y = model.getFourierSeries();
        double b = getStripWidth();
        double s = Y.getFunctionValue(y, m);
        double a = model.getModelLength();

        x = x / b;

        N.set(1-3*x*x + 2*x*x*x, 0, 0);
        N.set(x*b*(1-2*x+x*x), 0, 1);
        N.set(3*x*x - 2*x*x*x, 0, 2);
```

```
        N.set(x*b*(x*x - x), 0, 3);


        N.scale(s);
        return N;
    }
```

and

```
public Matrix getPlaneDisplacementShapeFunctionMatrix(double x, double y, int m) {
        Matrix N = Matrix.getMatrix(2, 4);
        Series Y = model.getFourierSeries();
        double b = getStripWidth();
        double s = Y.getFunctionValue(y, m);
        double s1 = Y.getVScalingValue(y, m);
        double a = model.getModelLength();

        x = x / b;

        N.set((1 - x) * s, 0, 0);
        N.set(0, 0, 1);
        N.set(x * s, 0, 2);
        N.set(0, 0, 3);

        N.set(0, 1, 0);
        N.set((1 - x) * a / Y.getMu_m(m) * s1, 1, 1);
        N.set(0, 1, 2);
        N.set(x * a / Y.getMu_m(m) * s1, 1, 3);

        return N;
    }
```

The nodal parameters for the Strip are returned by calling:

```
public Vector getParameterContributionVector(int m)
```

Where each parameter vector is in the form:

$$\begin{Bmatrix} u_{1[m]} \\ v_{1[m]} \\ w_{1[m]} \\ \theta_{1[m]} \\ u_{2[m]} \\ v_{2[m]} \\ w_{2[m]} \\ \theta_{2[m]} \end{Bmatrix}$$

and m such vectors exist. This vector is in fact simply the nodal parameters that are stored in the strip's nodes, arranged into a larger vector. If this vector is known, the displacements and stresses can be calculated according to equations 10, 29 and 30. In part 1 of Equation 10 the in-plane displacements are interpolated from the in-plane nodal parameters. The in-plane nodal parameters are picked from the strip's parameter vector and multiplied by their shape functions as follows:

```
1  public Vector getPlaneDisplacementVector(double localXCoordinate, double
       localYCoordinate) {
2          Vector f = Vector.getVector(2);
3          Vector param = Vector.getVector(4);
4
5          Matrix Nplane = Matrix.getMatrix(2, 4);
6
7          for (int m = 0; m < model.getFourierTerms(); m++) {
8              param.clear();
9              param.add(getRotationMatrix().transpose().multiply(
       getParameterContributionVector(m)).get(0), 0);
10             param.add(getRotationMatrix().transpose().multiply(
       getParameterContributionVector(m)).get(1), 1);
11             param.add(getRotationMatrix().transpose().multiply(
       getParameterContributionVector(m)).get(4), 2);
12             param.add(getRotationMatrix().transpose().multiply(
       getParameterContributionVector(m)).get(5), 3);
13
14             Nplane = getPlaneDisplacementShapeFunctionMatrix(localXCoordinate,
       localYCoordinate, m + 1);
15             f.add(Nplane.multiply(param));
16         }
17         param.release();
18         Nplane.release();
19         return f; }
```

Similarly, the bending (out of plane) nodal paramaters are picked from the strip's parameter vector and multiplied by the shapefunctions corresponding to bending.

```
public Vector getBendingDisplacementVector(double localXCoordinate, double
    localYCoordinate) {
        Vector w = Vector.getVector(1);
        Vector param = Vector.getVector(4);
        double a = model.getModelLength();
        Matrix Nbend = Matrix.getMatrix(1, 4);

        for (int m = 0; m < model.getFourierTerms(); m++) {

            param.clear();

            param.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(2), 0);
            param.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(3), 1);
            param.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(6), 2);
            param.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(7), 3);

            Nbend = getBendingDisplacementShapeFunctionMatrix(localXCoordinate,
    localYCoordinate, m + 1);
            w.add(Nbend.multiply(param));

        }

        param.release();
        Nbend.release();

        return w;
    }
```

The strain matrices $[B_B^{[m]}]$ and $[B_M^{[m]}]$ are necessary to convert the nodal paramaters into stresses. Recall equations 27 and 28:

$$[B_B^{[m]}] = \begin{bmatrix} \frac{6}{b^2}(1-2\bar{x})Y_m & \frac{2}{b}(2-3\bar{x})Y_m & \frac{6}{b^2}(-1+2\bar{x})Y_m & \frac{2}{b}(-3\bar{x}+1)Y_m \\ -(1-3\bar{x}^2+2\bar{x}^3)Y_m'' & -x(1-2\bar{x}+\bar{x}^2)Y_m'' & -(3\bar{x}^2-2\bar{x}^3)Y_m'' & -x(\bar{x}^2-\bar{x})Y_m'' \\ \frac{2}{b}(-6\bar{x}+6\bar{x}^2)Y_m' & 2(1-4\bar{x}+3\bar{x}^2)Y_m' & \frac{2}{b}(6\bar{x}-6\bar{x}^2)Y_m' & 2(3\bar{x}^2-2\bar{x})Y_m' \end{bmatrix} \quad \text{(27 revisited)}$$

$$[B_M^{[m]}] = \begin{bmatrix} \frac{-1}{b}Y_m & 0 & \frac{1}{b}Y_m & 0 \\ 0 & (1-\bar{x})\frac{a}{\mu_m}Y_m'' & 0 & (\bar{x})\frac{a}{\mu_m}Y_m'' \\ (1-\bar{x})Y_m' & \frac{-1}{b}\frac{a}{\mu_m}Y_m' & (\bar{x})Y_m' & \frac{1}{b}\frac{a}{\mu_m}Y_m' \end{bmatrix} \quad \text{(28 revisited)}$$

These are calculated programmatically as follows:

```
public Matrix getBendingStrainMatrix(double x, double y, int m) {
        Matrix B = Matrix.getMatrix(3, 4);
        B.clear();
        Series Y = model.getFourierSeries();

        double b = getStripWidth();

        double s = Y.getFunctionValue(y, m);
        double s1 = Y.getFirstDerivativeValue(y, m);
        double s2 = Y.getSecondDerivativeValue(y, m);
        double xb = x / b;

        B.set((6.0 / (b * b)) * (1 - 2 * xb) * s, 0, 0);
        B.set((2.0 / b) * (2 - 3 * xb) * s, 0, 1);
        B.set((6.0 / (b * b)) * (-1 + 2 * xb) * s, 0, 2);
        B.set((2.0 / b) * (-3 * xb + 1) * s, 0, 3);

        B.set(-(1 - 3 * xb * xb + 2 * xb * xb * xb) * s2, 1, 0);
        B.set(-x * (1 - 2 * xb + xb * xb) * s2, 1, 1);
        B.set(-(3 * xb * xb - 2 * xb * xb * xb) * s2, 1, 2);
        B.set(-x * (xb * xb - xb) * s2, 1, 3);

        B.set((2.0 / b) * (-6.0 * xb + 6.0 * xb * xb) * s1, 2, 0);
        B.set(2.0 * (1 - 4 * xb + 3 * xb * xb) * s1, 2, 1);
        B.set((2.0 / b) * (6.0 * xb - 6.0 * xb * xb) * s1, 2, 2);
```

```
        B.set(2.0 * (3.0 * xb * xb - 2.0 * xb) * s1, 2, 3);

        return B;
    }
```

and

```
public Matrix getPlaneStrainMatrix(double x, double y, int m) {
        Matrix B = Matrix.getMatrix(3, 4);
        B.clear();
        Series Y = model.getFourierSeries();
        double a = model.getModelLength();

        double b = getStripWidth();

        double s = Y.getFunctionValue(y, m);
        double s1 = Y.getFirstDerivativeValue(y, m);
        double s2 = Y.getSecondDerivativeValue(y, m);
        x = x / b;

        B.set((-1.0 / b) * s, 0, 0);
        B.set((1.0 / b) * s, 0, 2);
        B.set((1 - x) * (a / Y.getMu_m(m)) * s2, 1, 1);
        B.set(x * (a / Y.getMu_m(m)) * s2, 1, 3);
        B.set((1 - x) * s2, 2, 0);
        B.set((-1.0 / b) * (a / Y.getMu_m(m)) * s1, 2, 1);
        B.set(x * s1, 2, 2);
        B.set((1.0 / b) * (a / Y.getMu_m(m)) * s1, 2, 3);

        return B;
    }
```

To calculate the stresses we still require the bending and membrane property matrices $[D_B]$ and $[D_M]$. Recall equations 31 and 32:

$$[D_B] = \begin{bmatrix} D_x & D_1 & 0 \\ D_1 & D_y & 0 \\ 0 & 0 & D_{xy} \end{bmatrix} \tag{31 revisited}$$

$$[D_M] = \begin{bmatrix} \frac{E_x}{1 - v_x v_y} & \frac{v_x E_y}{1 - v_x v_y} & 0 \\ \frac{v_x E_y}{1 - v_x v_y} & \frac{E_x}{1 - v_x v_y} & 0 \\ 0 & 0 & G_{xy} \end{bmatrix} \tag{32 revisited}$$

The matrices are calculated by the methods:

```
public Matrix getPlanePropertyMatrix() {

        Matrix D = Matrix.getMatrix(3, 3);
        Material mat = model.getModelMaterial();
        double Ex = mat.getEx();
        double Ey = mat.getEy();
        double vx = mat.getVx();
        double vy = mat.getVy();
        double G = mat.getG();
        double E1 = Ex / (1 - vx * vy);
        double E2 = Ey / (1 - vx * vy);
        D.set(E1, 0, 0);
        D.set(vx * E2, 0, 1);
        D.set(vx * E2, 1, 0);
        D.set(E2, 1, 1);
        D.set(G, 2, 2);
        return D;
    }
```

and

```
public Matrix getBendingPropertyMatrix() {
        Matrix D = Matrix.getMatrix(3, 3);
        D.clear();
        Material mat = model.getModelMaterial();
        double t = this.t.doubleValue();

        double Ex = mat.getEx();
        double Ey = mat.getEy();
        double vx = mat.getVx();
        double vy = mat.getVy();
        double G = mat.getG();

        double Dx = (Ex * t * t * t) / (12.0 * (1 - vx * vy));
        double Dy = (Ey * t * t * t) / (12.0 * (1 - vx * vy));
        double D1 = (vx * Ey * t * t * t) / (12.0 * (1 - vx * vy));
        double Dxy = G * t * t * t / 12.0;

        D.set(Dx, 0, 0);
        D.set(D1, 0, 1);

        D.set(D1, 1, 0);
        D.set(Dy, 1, 1);

```

```
        D.set(Dxy, 2, 2);

        return D;
    }
```

Now that we have methods for the calculation of $[B_M^{[m]}]$, $[B_B^{[m]}]$, $[D_B]$ and $[D_M]$, the stresses can be found. Recall equations 29 and 30:

$$\{\sigma_B\} = \begin{bmatrix} M_x \\ M_y \\ M_x y \end{bmatrix} = [D_B]\{\epsilon_B\} = [D_B] \sum_{m=1}^{q} [B_B^{[m]}]\{d_w^{[m]}\} \qquad \text{(29 revisited)}$$

$$\{\sigma_M\} = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix} = [D_M]\{\epsilon_M\} = [D_M] \sum_{m=1}^{q} [B_M^{[m]}]\{d_{uv}^{[m]}\} \qquad \text{(30 revisited)}$$

These are calculated programmatically as follows:

```
public Vector getBendingStressVector(double localXCoordinate, double localYCoordinate)
    {
        Vector ub = Vector.getVector(4);

        Vector strain = Vector.getVector(3);
        strain.clear();

        for (int m = 0; m < model.getFourierTerms(); m++) {
            ub.clear();

            ub.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(2), 0);
            ub.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(3), 1);
            ub.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(6), 2);
            ub.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(7), 3);

            Matrix B = getBendingStrainMatrix(localXCoordinate, localYCoordinate, m +
    1);

            strain.add(B.multiply(ub));

            B.release();
```

```
        }

        ub.release();
        return getBendingPropertyMatrix().multiply(strain);
    }
```

and

```
public Vector getPlaneStressVector(double localXCoordinate, double localYCoordinate) {
        Vector um = Vector.getVector(4);

        Vector strain = Vector.getVector(3);
        strain.clear();

        for (int m = 0; m < model.getFourierTerms(); m++) {
            um.clear();

            um.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(0), 0);
            um.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(1), 1);
            um.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(4), 2);
            um.add(getRotationMatrix().transpose().multiply(
    getParameterContributionVector(m)).get(5), 3);

            Matrix B = getPlaneStrainMatrix(localXCoordinate, localYCoordinate, m + 1);

            strain.add(B.multiply(um));

            B.release();
        }

        um.release();
        return getPlanePropertyMatrix().multiply(strain);
    }
```

Note the rotation of the parameter vector done in the source code that is not present in equations 29 and 30. This is done because the results are reported in strip local coordinates rather than global (system) coordinates.

The strip object can also calculate a few of its cross sectional properties. The cross-sectional area, which is just the strips width multiplied by its thickness is given by:

```
public double getCrossSectionalArea() {
```

```
        return getStripThickness() * getStripWidth();
    }
```

The centroid coordinates of the cross section is just the midpoint's coordinates between the strips nodes:

```
public Point2D.Double getCrossSectionalCentroid() {
        return new Point2D.Double((node1.getXCoord() + node2.getXCoord()) / 2.0, (node1
    .getZCoord() + node2.getZCoord()) / 2.0);
    }
```

The horizontal and vertical distances from the origin to the strip's centroid are called $\bar{x}$ and $\bar{z}$ respectively. They are given by:

```
public double getXBar() {
        return getCrossSectionalCentroid().getX();
    }
```

and

```
public double getZBar() {
        return getCrossSectionalCentroid().getY();
    }
```

The moment of inertia for a rectangle, rotated by angle θ, with respect to its centroid axis a-a is given by:

$$I_{aa} = \frac{bh^3}{12}(h^2 \cos\theta^2 + b^2 \sin\theta^2) \tag{52}$$

Equation 52 was implemented to calculate the moment of inertia of a strip around its centroid x-x and z-z axis respectively.

```
public double getIxx()
    {
        double cos = Math.cos(getStripAngle());
        double sin = Math.sin(getStripAngle());

        double b = getStripWidth();
        double d = getStripThickness();


        return (b*d/12.0)*(d*d*cos*cos + b*b*sin*sin);
    }
```

```
public double getIzz() // Iyy in redbook
    {
        double cos = Math.cos(getStripAngle());
        double sin = Math.sin(getStripAngle());

        double d = getStripWidth();
        double b = getStripThickness();


        return (b*d/12.0)*(d*d*cos*cos + b*b*sin*sin);
    }
```

The product of inertia is given by:

$$I_{xz} = -\tan 2\theta \frac{(I_{xx} - I_{zz})}{2} \tag{53}$$

And is calculated programmatically as follows:

```
public double getIxz() // Product of inertia
    {
        return -Math.tan(2*getStripAngle())*(getIxx() - getIzz())/2.0;
    }
```

6.1.6 Model

A Model object represents a single physical member consisting of a set of Nodes and a set of Strips. It has length, material, end conditions determined by the series function used, a maximum number of terms and a single buckling load.

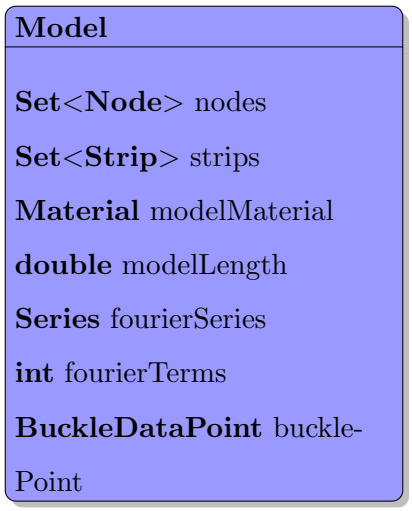


Figure 17: Object diagram for a Model object.

The Nodes and Strips that make up a Model can be added, retrieved and removed by the following methods:

```
public void addNode(Node n)
public void addStrip(Strip s)
public Node getNode(int Id)
public void removeNode(Node n)
public void removeStrip(Strip n)
```

The model's material can be retrieved or changed by calling:

```
public void setModelMaterial(Material modelMaterial)
public double getModelLength()
```

The length can be retrieved or changed by calling:

```
public double getModelLength()
public void setModelLength(double modelLength)
```

The number of m-terms can be retrieved or changed by calling:

```
public void setFourierTerms(int fourierTerms)
public int getFourierTerms()
```

The series function can be retrieved or changed by calling:

```
public Series getFourierSeries()
public void setFourierSeries(Series fourierSeries)
```

For buckling problems the maximum allowable centerline stress may be retrieved or changed by calling:

```
public double getAllowableStress()
public void setAllowableStress(double stress)
```

The model object should have the ability to calculate some of its cross-sectional properties. The area for the entire model is just the sum of its individual strips. This is done programmatically as follows:

```
public double getCrossSectionalArea() {
        double A = 0;

        for (Strip s : strips) {
            A += s.getCrossSectionalArea();
        }

        return A;
    }
```

The moment of inertia for a cross section consisting of n parts, with respect to its centroid axis a-a is given by:

$$I_{aa} = \sum_{i=0}^{n} I'_n + A_n d_n^2 \tag{54}$$

This is done programmatically for the x-x and z-z axis respsectively:

```
public double getIxx() {

        //I = sum(Ixx +Ad^2)
        double I = 0;
        double zbar = getCentroidZ();


        for (Strip s : strips) {

            double d = zbar - s.getZBar();

            I = I + s.getIxx() + s.getCrossSectionalArea() * d * d;
        }

        return I;
    }
```

```
public double getIzz() {

        //I = sum(Izz +Ad^2)
        double I = 0;
        double xbar = getCentroidX();


        for (Strip s : strips) {

            double d = xbar - s.getXBar();

            I = I + s.getIzz() + s.getCrossSectionalArea() * d * d;
        }

        return I;
    }
```

The product of inertia for a cross section consisting of n parts, with respect to its x-z is given by:

$$I_{zx} = \sum_{i=0}^{n} I'_n + A_n dx_n dz_n \tag{55}$$

which is calculated programmatically as follows:

```
public double getIxz() // product of inertia
    {

        // I = Ixz + Adxdy
        double I = 0;
        double zbar = getCentroidZ();
        double xbar = getCentroidX();

        for (Strip s : strips) {
            double dz = zbar - s.getZBar();
            double dx = xbar - s.getXBar();

            I = I + s.getIxz() + s.getCrossSectionalArea() * dx * dz;
        }

        return I;

    }
```

The angle of the principal axis can be calculated with:

$$\alpha = 0.5 \arctan\left(\frac{-2I_{xz}}{I_{xx} - I_{zz}}\right) \tag{56}$$

and can be implemented programmatically as follows:

```
public double getPrincipalAxisAngle() {

        double Ixz = getIxz();
        double Ixx = getIxx();
        double Izz = getIzz();

        if(Ixz == 0)
        {
            return 0.0;
        }
        return Math.atan((-Ixz * 2.0) / (Ixx - Izz)) / 2.0;
    }
```

With the principal axis angle known, we can calculate the prinicpal moments of inertia using:

$$I_{xxp} = \frac{I_{xx} + I_{zz}}{2} + \frac{I_{xx} - I_{zz}}{2}\cos 2\alpha - I_{xz}\sin 2\alpha \tag{57}$$

and

$$I_{zzp} = \frac{I_{xx} + I_{zz}}{2} - \frac{I_{xx} - I_{zz}}{2}\cos 2\alpha + I_{xz}\sin 2\alpha \tag{58}$$

With the source code being:

```
public double getIxxPrincipal() {

        double Ixz = getIxz();
        double Ixx = getIxx();
        double Izz = getIzz();
        double theta = getPrincipalAxisAngle();

        return ((Ixx + Izz) / 2.0) + ((Ixx - Izz) / 2.0) * Math.cos(2 * theta) - Ixz *
    Math.sin(2 * theta);

    }
```

and

```
public double getIzzPrincipal() {

        double Ixz = getIxz();
```

```
4         double Ixx = getIxx();
5         double Izz = getIzz();
6         double theta = getPrincipalAxisAngle();
7
8         return ((Ixx + Izz) / 2.0) - ((Ixx - Izz) / 2.0) * Math.cos(2 * theta) + Ixz *
      Math.sin(2 * theta);
9     }
```

6.1.7 Assembler

Individual strips are not of much use unless they are assembled into a system. For the simply supported case, the assembly process is similar to that of the Finite Element method, except that the process has to be repeated for each m-term.

Figure 18: Object diagram for an Assembler object.

The assembly process is done by use of a topology matrix. For illustration purposes we consider two strips, each having only one numbered degree of freedom at each node:

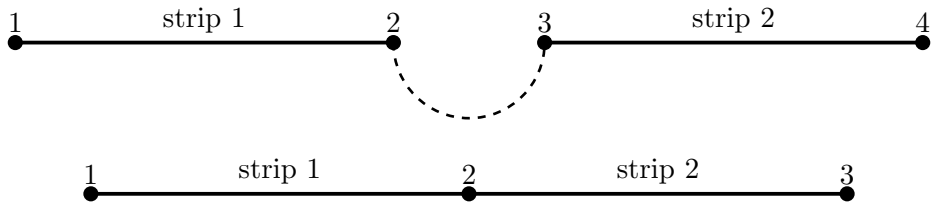


Figure 19: The assembly of two finite strips.

For the assembly shown in Figure 19, we have a local index numbering of {1,2,3,4} and in the global configuration we have {1,2,3}. To map each degree of freedom in the local configuration to the relevant degree of freedom in the assembly we create a local to global configuration array that is as follows:

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 2 \\ 4 & 3 \end{bmatrix}$$

From this local to global configuration array we can set up the topology matrix T by taking an empty matrix of p rows by q columns, where p is the number of degrees of freedom in the local configuration and q the number of degrees of freedom in the global configuration. Then, the elements of this matrix are set equal to 1 at the indices dictated by each row in the local to global configuration array. For

the system shown in Figure 19, this would result in a 4 by 3 matrix with 1 values at (1,1) , (2,2) , (3,2) and (4,3).

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To get the stiffness matrix for the system, we assemble a matrix K_T with the stiffness matrices of all strips stacked on the diagonal. The system stiffness matrix K_s is then found by:

$$[K_s] = [T]^T [K_T][T] \tag{59}$$

The process of setting the values of $[T]$ is done in the constructor of Assembler as follows:

```
public Assembler(List<Strip> strips, int nrOfGlobalDOF, int[][]
    localToGlobalConfNumbering) {

        localDOF = localToGlobalConfNumbering.length;
        globalDOF = nrOfGlobalDOF;
        nrOfElements = strips.size();

        T = Matrix.getMatrix(localDOF, globalDOF);
        T.clear();
        for (int i = 0; i < localDOF; i++) {

            T.set(1, localToGlobalConfNumbering[i][0], localToGlobalConfNumbering[i
    ][1]);
        }

        this.strips = strips;

    }
```

The source code for assembling the system stiffness matrix is shown below:

```
public Matrix getK(int m) {
        Matrix Kt = Matrix.getMatrix(localDOF, localDOF);
        Kt.clear();

        for (int i = 0; i < nrOfElements; i++) {

            int[] indices = new int[strips.get(i).getStiffnessMatrix(m , m).cols()];
            for (int j = 0; j < indices.length; j++) {
                indices[j] = i * indices.length + j;
            }

            Kt.addSubmatrix(strips.get(i).getRotatedStiffnessMatrix(m, m), indices);
        }

        Matrix K = T.transpose().multiply(Kt).multiply(T);

        return K;
    }
```

The assembly of the system load vector is dealt with in a similar fasion. The individual load vectors of each element is stacked in a vector $\{F_T\}$. The system load vector $\{F_s\}$ is then found using:

$$\{F_s\} = [T]^T \{F_T\} \tag{60}$$

and is done programmatically as follows:

```
public Vector getF(int m)
    {
        Vector Ft = Vector.getVector(localDOF);
        Ft.clear();

        for (int i = 0; i < nrOfElements; i++)
        {
            for (int j = 0; j < strips.get(i).getLoadVector(m).size(); j++)
            {
                Ft.set(strips.get(i).getRotatedLoadVector(m).get(j), j + i*strips.get(i
    ).getLoadVector(m).size());
            }

        }
        Vector F = T.transpose().multiply(Ft);

        return F;
    }
```

6.1.8 PartitionedSystem

In a particular system, some of the nodal degrees of freedom may be prescribed. When considering such a system, the system equation is partitioned so the free degrees of freedom may be solved separately. The PartitionedSystem object provides us with the functionality needed to solve a partitioned system.

$$\begin{bmatrix} [K_{ff}] & [K_{fp}] \\ [K_{pf}] & [K_{pp}] \end{bmatrix} \begin{Bmatrix} \{U_f\} \\ \{U_p\} \end{Bmatrix} = \begin{Bmatrix} \{P_f\} \\ \{P_p\} \end{Bmatrix} \tag{61}$$

In which $\{U_f\}$ is a vector containing the free degrees of freedom and $\{U_p\}$ contains the prescribed degrees of freedom.

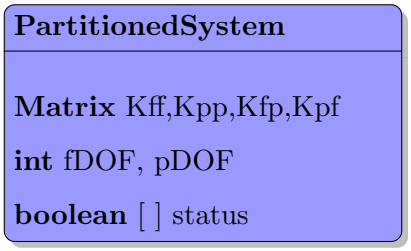


Figure 20: Object diagram for an Assembler object.

The status vector indicates which degrees of freedom are free and which are prescribed and is used to identify the partitioned matrices $[K_{ff}], [K_{fp}], [K_{pp}]$ and $[K_{pf}]$ as well as $\{P_f\}$ and $\{P_p\}$. The submatrices of the system stiffness matrix, or the free part of the load vector may be found by calling one of the following methods:

```
1 public Matrix getKpp()
2 public Matrix getKff()
3 public Matrix getKpf()
4 public Matrix getKfp()
5 public Vector getWf(Vector Ws)
```

Once the neccesary matrices and vectors are known $\{U_f\}$ may be solved by using:

$$\{U_f\} = [K_{ff}]^{-1}(-[K_{fp}]\{U_p\} + \{P_f\}) \tag{62}$$

For this process performing the inverse directly is quite inefficient so $[K_{ff}]$ is decomposed using Cholesky decomposition. The system may then be solved by performing backward and forward sweeps.

6.1.9 Cholesky

Cholesky is a utility class that provides the functionality to solve a sytem of equations with the form:

$$[A]\{X\} = \{Y\} \tag{63}$$

where $[A]$ and $\{Y\}$ are known and the solution for $\{X\}$ is required without performing the inversion of $[A]$ directly. The Cholesky class provides this functionality by including a method that returns the unknown $\{X\}$ given $[A]$ and $\{Y\}$.

```
public Vector getX(Matrix A , Vector y)
```

6.1.10 SystemEquation

Now that the basic functionality of a full FSM implementation exists, a class is required that performs the assembly process, calculates the unknown nodal displacements and stores the nodal parameters in the node objects for later processing. The SystemEquation class provides this functionality.

The assembly is done from the nodes and strips that are stored in the model object passed to the constructor:

```
public SystemEquation(Model model)
```

The parameter vectors are then calculated by finding a solution to the system equation for each required m-term, and stored in the model's nodes.

```
public void computeParameterVector()
```

6.1.11 BucklingEquation

The BucklingEquation class performs the same task as the SystemEquation class, but for buckling problems instead of static analysis. The assembly is done from the nodes and strips that are stored in the model object passed to the constructor:

```
public BucklingEquation(Model model)
```

The buckling data is then computed and return in the form of a BuclingDataPoint object. To generate a complete buckling curve, the method should be called for a range of models with different lengths.

```
public BucklingDataPoint getBucklingData()
```

6.2 DSM Implementation

6.2.1 Requirements

As stated earlier, the Direct Strength Method has two main requirements. Firstly, accurate modelling of elastic buckling behavior and secondly, the ability to separate buckling modes. By completing the implementation described in the previous section, we have fulfilled both these requirements. What we require now is a means by wich to programmatically do the DSM calculations of Section 4.3. The DSM equations themselves are quite simple and therefore are not hard to implement in JAVA.

Having a closer look at the equations we can identify 2 main scenarios with 4 steps each. At step 1 there exist 2 scenarios as well.

Scenario 1 : Axial behaviour

1. Determine nominal axial strength for global buckling, P_{ne}.

   - Scenario A - The member is laterally supported.

$$P_{ne} = P_y \tag{64}$$

   - Scenario B - The member is not laterally supported.

$$P_{ne} = \begin{cases} (0.658^{{\lambda_c}^2})P_y & \text{if } \lambda_c \leq 1.5, \\ \dfrac{0.877}{{\lambda_c}^2}P_y & \text{if } \lambda_c < 1.5 \end{cases} \tag{45 revisited}$$

2. Determine nominal axial strength for local buckling, P_{nl}

$$P_{nl} = \begin{cases} \left[1 - 0.15(P_{crl}/P_{ne})^{0.4}\right]\left(\dfrac{P_{crl}}{P_{ne}}\right)^{0.4} P_{ne} & \text{if } \lambda_l > 0.776, \\ P_{ne} & \text{if } \lambda_l \leq 0.776 \end{cases} \tag{46 revisited}$$

3. Determine nominal axial strength for distortional buckling, P_{nd}

$$P_{nd} = \begin{cases} \left[1 - 0.25(P_{crd}/P_y)^{0.6}\right]\left(\dfrac{P_{crd}}{P_y}\right)^{0.6} P_y & \text{if } \lambda_d > 0.561, \\ P_y & \text{if } \lambda_d \leq 0.561 \end{cases} \tag{47 revisited}$$

4. Determine the nominal axial strength.

$$P_n = \min(P_{ne}, P_{nl}, P_{nd}) \tag{44 revisited}$$

Scenario 2 : Flexural behaviour

1. Determine nominal flexural strength for global buckling, M_{ne}

   - Scenario A - The member is laterally supported.

$$M_{ne} = M_y \tag{65}$$

   - Scenario B - The member is not laterally supported.

$$M_{ne} = \begin{cases} M_{cre} & \text{if } M_{cre} < 0.56M_y, \\ \frac{10}{9}M_y\left(1 - \frac{10M_y}{36M_{cre}}\right) & \text{if } 2.78My \geq M_{cre} \geq 0.56M_y \\ M_y & \text{if } M_{cre} > 2.78M_y \end{cases} \tag{49 revisited}$$

2. Determine nominal flexural strength for local buckling, M_{nl}

$$M_{nl} = \begin{cases} \left[1 - 0.15(M_{crl}/M_{ne})^{0.4}\right]\left(\frac{M_{crl}}{M_{ne}}\right)^{0.4} M_{ne} & \text{if } \lambda_l > 0.776, \\ M_{ne} & \text{if } \lambda_l \leq 0.776 \end{cases} \tag{50 revisited}$$

3. Determine nominal flexural strength for distortional buckling, M_{nd}

$$M_{nd} = \begin{cases} \left[1 - 0.22(M_{crd}/M_y)^{0.5}\right]\left(\frac{M_{crd}}{M_y}\right)^{0.5} M_y & \text{if } \lambda_d > 0.673, \\ M_y & \text{if } \lambda_d \leq 0.673 \end{cases} \tag{51 revisited}$$

4. Determine the nominal flexural strength.

$$M_n = \min(M_{ne}, M_{nl}, M_{nd}) \tag{48 revisited}$$

6.2.2 JAVA Object Definitions

To perform the DSM calculations programmatically we only need one JAVA class that incorporates the steps and functionality described in the previous section. This class will be called DSMCalcs.

As the program has no intelligence to distinguish between axial and flexural load, the user will have to specify the type of load that was applied. The correct methods in DSMCalcs will then have to be called based on this decision. So, for scenario 1 of section 6.2.1 the methods corresponding to axial load will be called. First, the input values from the Finite Strip Analysis should be set by calling:

```
1 public void setPy(double Py)
2 public void setPcrl(double Pcrl)
3 public void setPcrd(double Pcrd)
4 public void setPcre(double Pcre)
5 public void setPhiC(double phiC)
```

then the nominal compressive strength strength may be calculated by calling:

```
public double getNominalCompressiveStrength(boolean braced) {
        double Pne = 0;
        double Pnl = 0;
        double Pnd = 0;

        //calcArea.appendText("Pcre = " + Pcre + "\n");
        //calcArea.appendText("Cb = " + Cb + "\n");
        //Pcre = Cb * Pcre;
        //calcArea.appendText("Mcre := Cb Mcre = " + Mcre + "\n");

        Pne = getPne(braced);

        Pnl = getPnl(Pne);
        Pnd = getPnd();

        calcArea.appendText("----------------------------------------------------------" + "\n"
    );
        calcArea.appendText("Predicted compressive strength per DSM 1.2" + "\n");
        calcArea.appendText("----------------------------------------------------------" + "\n"
    );
        double Pn = Math.min(Math.min(Pne, Pnl), Pnd);

        calcArea.appendText("Pne = " + Pne + "\n");
        calcArea.appendText("Pnl = " + Pnl + "\n");
        calcArea.appendText("Pnd = " + Pnd + "\n");
        calcArea.appendText("per DSM 1.2.1, Pn is the minimum of Pne, Pnl, Pnd.\n");
        calcArea.appendText("Pn = " + Pn + "\n");

        calcArea.appendText("LRFD :   c = " + phiC + "\n");
        calcArea.appendText("  b  Mn = " + phiC * Pn + "\n");

        return Pn;
    }
```

where the boolean "braced" will tell the program whether or not the member is laterally supported. For scenario 2 the methods corresponding to flexural load should be called. First, the input values from the Finite Strip analysis should be set by calling:

```
public void setMy(double My)
public void setMcrl(double Mcrl)
public void setMcrd(double Mcrd)
public void setMcre(double Mcre)
public void setCb(double Cb)
```

```
public void setPhiB(double phiB)
```

then the nominal flexural strength can be calculated by calling:

```
public double getNominalFlexuralStrength(boolean braced) {

        double Mne = 0;
        double Mnl = 0;
        double Mnd = 0;

        calcArea.appendText("Mcre = " + Mcre + "\n");
        calcArea.appendText("Cb = " + Cb + "\n");
        Mcre = Cb * Mcre;
        calcArea.appendText("Mcre := Cb Mcre = " + Mcre + "\n");

        Mne = getMne(braced);

        Mnl = getMnl(Mne);
        Mnd = getMnd();

        calcArea.appendText("-------------------------------------------------------" + "\n"
    );
        calcArea.appendText("Predicted flexural strength per DSM 1.3" + "\n");
        calcArea.appendText("-------------------------------------------------------" + "\n"
    );
        double Mn = Math.min(Math.min(Mne, Mnl), Mnd);

        calcArea.appendText("Mne = " + Mne + "\n");
        calcArea.appendText("Mnl = " + Mnl + "\n");
        calcArea.appendText("Mnd = " + Mnd + "\n");
        calcArea.appendText("per DSM 1.2.2, Mn is the minimum of Mne, Mnl, Mnd.\n");
        calcArea.appendText("Mn = " + Mn + "\n");

        calcArea.appendText("LRFD :   b = " + phiB + "\n");
        calcArea.appendText("  b  Mn = " + phiB * Mn + "\n");

        return Mn;
    }
```

7 Verification of implementation

7.1 Simple Bending

7.1.1 Description

The simple bending problem is chosen to verify the integrity of in-plane stresses developed in the plane-stress strip as well as the bending moment developed in the bending strip during bending. Two models are used, the first (Figure 21) consisting of ten equally spaced plane-stress strips and the second (Figure 22) consisting only of one bending strip. Both models are chosen to represent a beam profile of width $b = 100$, height $h = 100$ and length $L = 4000$. Note that for illustrative purposes figures are not drawn to scale.

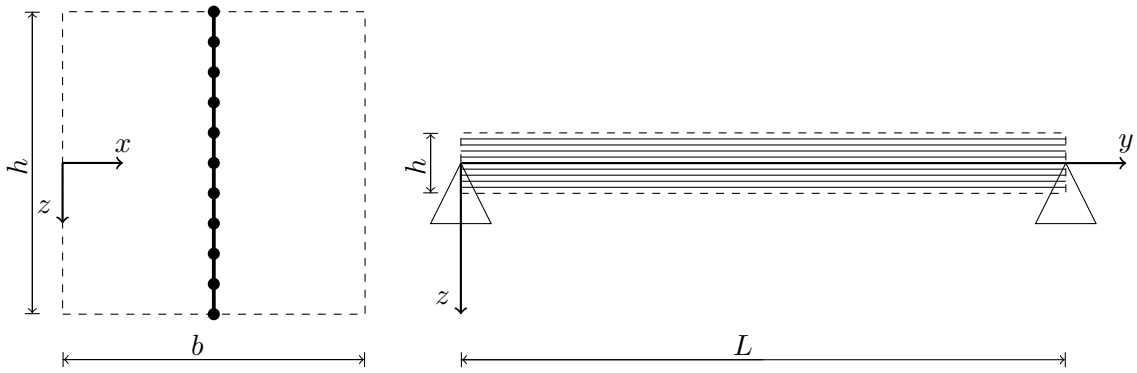


Figure 21: Ten plane-stress strips representing a simply-supported beam in bending.

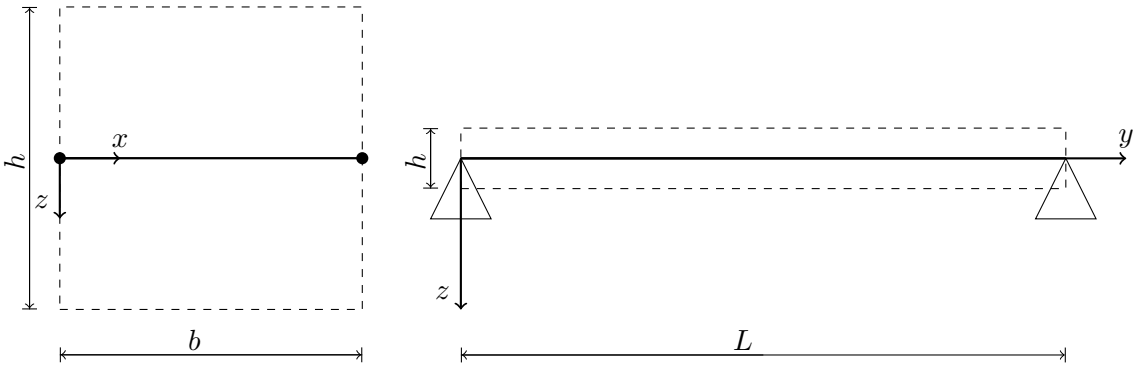


Figure 22: Single bending strip representation of a simply-supported beam.

A uniformly distributed load of $w = 1.0$ is applied to both models to cause bending in the z-direction. According to Euler's beam theory, $M_{max} = wL^2/8 = 2 \times 10^6$ is the expected value for the moment at $L/2$. Note that the stress values obtained from bending strips are in fact moment per unit length i.e. $N.m/m$, whereas each plane-stress strip produces a stress i.e. N/m^2 in its local y-direction. These

stresses then need to be converted to a resulting moment for comparison either by multiplying by the section width (bending strip) or integrating over the section height (plane-stress strip).

7.1.2 Results

After creating each model and specifying the load, it is possible to obtain the necessary stresses and plot the distributions. Figures 23 and 24 are generated output for each model by using a total number of longitudinal terms $r = 1$.

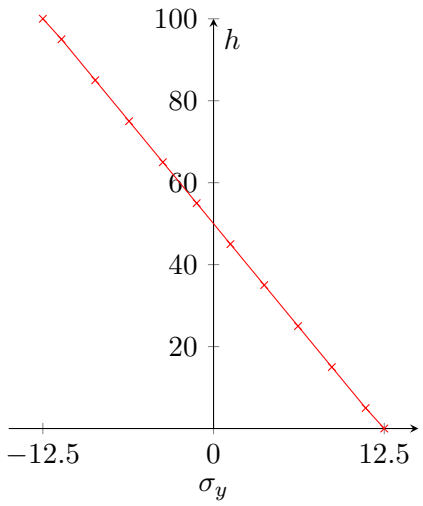


Figure 23: In-plane stress at $L/2$.

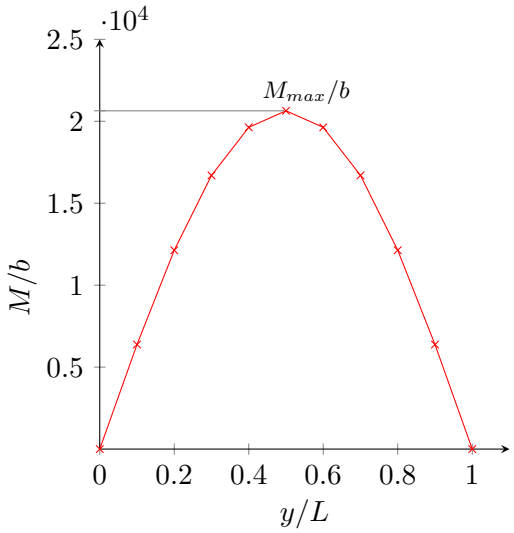


Figure 24: Bending moment along L.

By inspection, it is clear that the stress distributions are as expected, or at least the shape seems correct. Calculating the resulting moment from the in-plane stress can be done by use of $\sigma_y = My/I$. This gives a value of $M_{max} = 2.083 \times 10^6$ for the plane-stress strips. The obtained value for $M_{max}/b = 20639.23$, multiplying by b gives $M_{max} = 2.064 \times 10^6$ for the bending strip. While both of these answers are fairly close to the theoretical value of $M_{max} = wL^2/8 = 2 \times 10^6$, they can be further improved by increasing the number of longitudinal terms.

To confirm that an increased number of longitudinal terms does indeed increase calculation accuracy the simulation is run again, but this time with the longitudinal terms increased to $r = 10$.

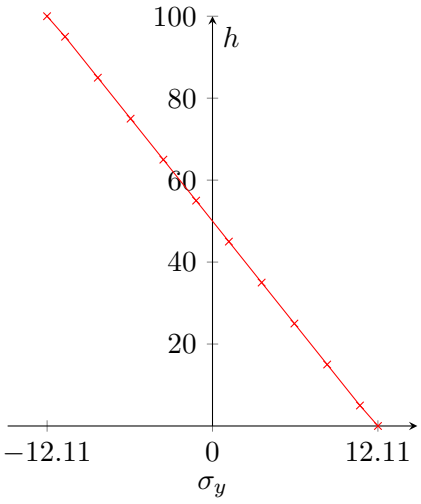


Figure 25: In-plane stress at $L/2$.

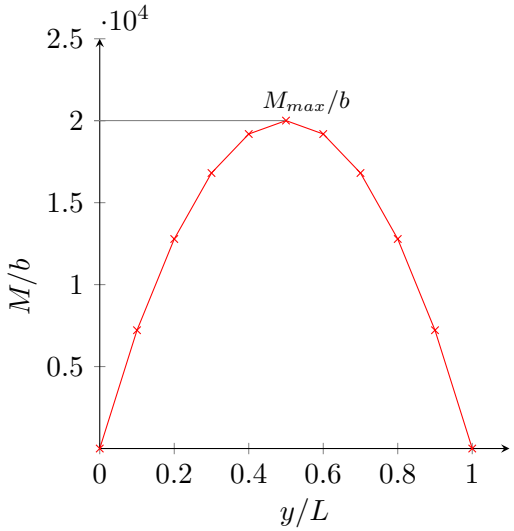


Figure 26: Bending moment along L.

We can see that $\sigma_y = 12.11$ at the outer edge, giving $M_{max} = 2.018 \times 10^6$ for the plane-stress strips. The obtained value for $M_{max}/b = 20008.31$ in Figure 26, multiplying by b gives $M_{max} = 2.00 \times 10^6$.

7.1.3 Discussion

In this section, a sample problem was chosen to verify the integrity of in-plane stresses developed in the plane-stress strip as well as the bending moment developed in the bending strip during bending. Two models were used, the first (Figure 21) consisting of ten equally spaced plane-stress strips and the second (Figure 22) consisting only of one bending strip. Both models were chosen to represent the same beam profile. Both models were subjected to a uniformly distributed load to cause bending in the z-direction.

The maximum bending moment was calculated for each model and it was found that for a total number of longitudinal terms equal to 1, both models produced acceptable results, however, the single bending strip performed slightly better. This is because a high number of plane stress strips are needed to model the varying stress along the cross section while the single bending strip can represent this varying stress distribution with a single moment value. Increasing the total number of longitudinal terms to $r = 10$, of course, increased the accuracy of the result.

7.2 Plane stress - Deep Beam

7.2.1 Description

In order to verify the integrity of the program, two examples from Cheung [1] are repeated. The first deals with the analysis of a simply supported square deep beam. The second example deals with the same square deep beam, but with clamped ends. Both models are analysed with a width $b = 50$, height $h = 100$ and length $L = 100$ (Figure 27).

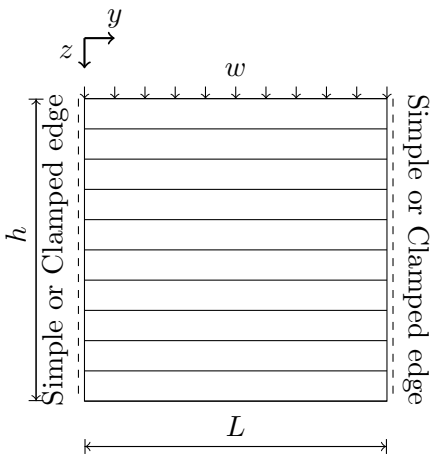


Figure 27: FSM idealisation.

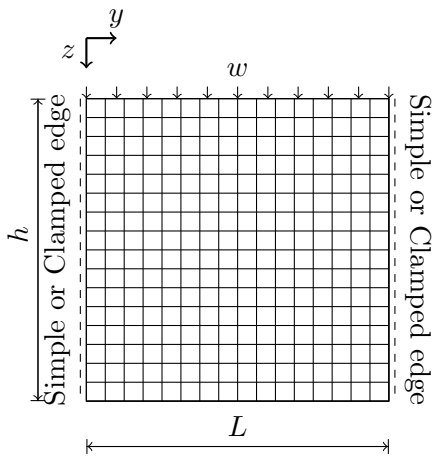


Figure 28: FEM idealisation.

A uniformly distributed load of $w = 1.0$ is applied to both models to cause bending in the z-direction. Note that, because h/L is not small as in Problem 1, Euler's beam equations no longer apply. A non-linear stress distribution through the thickness of the beam is expected. Consequently, the stress distribution computed by the program at $L/2$ needs to be verified against a FEM model (Figure 28).

7.2.2 Results

Two finite-strip models are created, both with 10 equally-spaced plane-stress strips and a total of 10 longitudinal terms (Figure 27). The first finite-strip model is analysed with simply-supported boundary conditions and the second with clamped boundary conditions. For comparison, two finite-element models are created in ABAQUS using 256 shell elements (16×16). The first with the vertical displacement fixed at both edges to simulate simply-supported boundary conditions and the second with vertical and horizontal displacement as well as rotation fixed to simulate clamped end-conditions. A uniformly distributed line load, $w = 1.0$ is applied to the top of each model and the horizontal and vertical stress distributions are plotted at $L/2$.

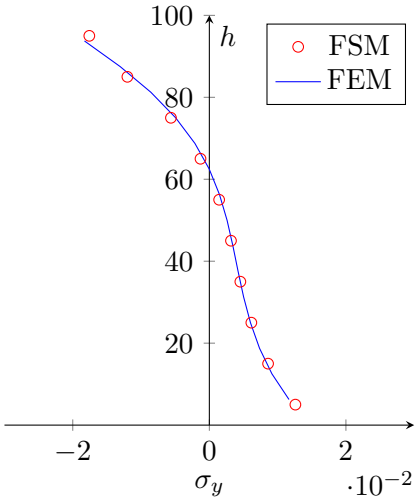


Figure 29: σ_y for SS deep beam at $L/2$.

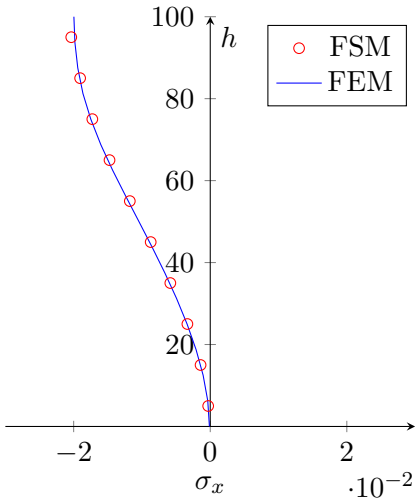


Figure 30: σ_x for SS deep beam at $L/2$.

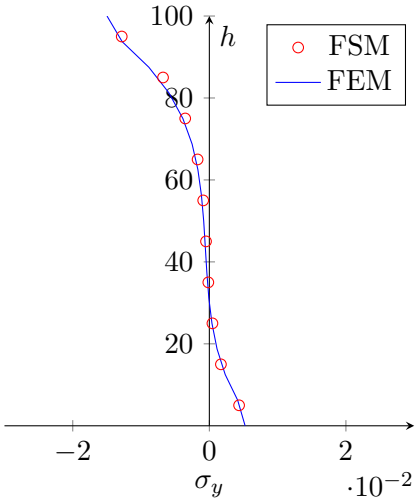


Figure 31: σ_y for CC deep beam at $L/2$.

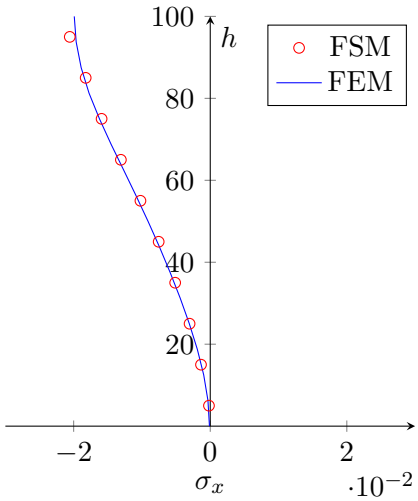


Figure 32: σ_x for CC deep beam at $L/2$.

From figures 29 to 32 it is clear that there is a good correlation between the FEM and FSM results for both simple and clamped boundary conditions.

7.2.3 Discussion

In this section the longitudinal and transverse stresses, along the cross-section of a deep beam, were determined using both FSM and FEM. From figures 29 to 32 it is clear that there is a good correlation between the FEM and FSM results for both simple and clamped boundary conditions. These results both verify the accuracy of FSM and suggests that the theory has been correctly implemented.

7.3 Validation studies for buckling solution

To validate the output from the Finite Strip implementation for buckling problems, comparisons are made to classical plate buckling solutions, shell finite element models and the widely used Finite Strip implementation namely CUFSM.

The buckling stress σ_{cr} of a plate is given by:

$$\sigma_{cr} = k\frac{\pi^2 E}{12(1-v^2)}\left(\frac{t}{b}\right)^2 \tag{66}$$

where k is the plate buckling coefficient and is dependent on loading and boundary conditions E and v are material properties, t is the plate thickness and b the plate width.

7.3.1 Description

In a paper by Z. Li [2] the plate buckling coefficient is determined for a 63.5mm wide, 1.27mm thick plate, with E=203000 MPa and v=0.3. In this section, the plate buckling coefficient is determined for the same plate with two end boundary conditions namely simple-simple (S-S) and clamped-clamped (C-C) with longitudinal boundary conditions simple-simple (s-s). The results are generated using the Finite Strip implementation developed in section 6 and compared to classical plate buckling solutions, shell finite element models and the widely used Finite Strip implementation CUFSM.

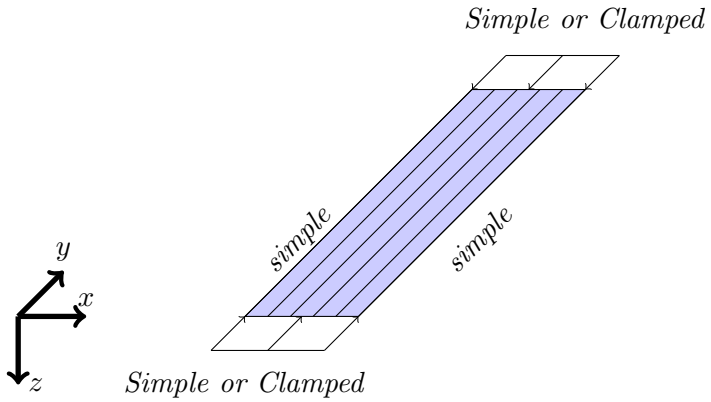


Figure 33: Plate boundary conditions under consideration.

The plate shown above was modelled using 10 strip elements each with a width of 6.35mm making up a total plate with a width of 63.5mm and given a thickness of 1.27mm. The plate was given a length of 1000mm to ensure that data is generated up to a length to width ratio of at least 5. Two buckling curves were generated with 100 plot points each. One for the model with simple boundary conditions all around and the other with one pair of edges clamped and simple boundary conditions along the longitudinal edges.

7.3.2 Results

The programs' output provides us with the σ_{cr} values for a range of physical lengths. The σ_{cr} values can then be used together with the material properties and dimensions to determine the plate coefficient k at different physical lengths using Equation 66.

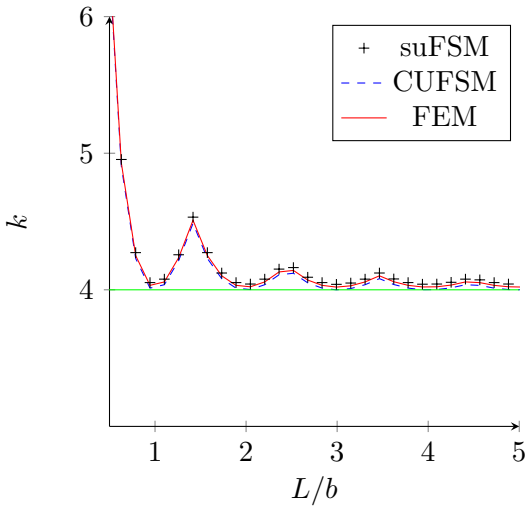


Figure 34: Buckling coefficients, S-S s-s.

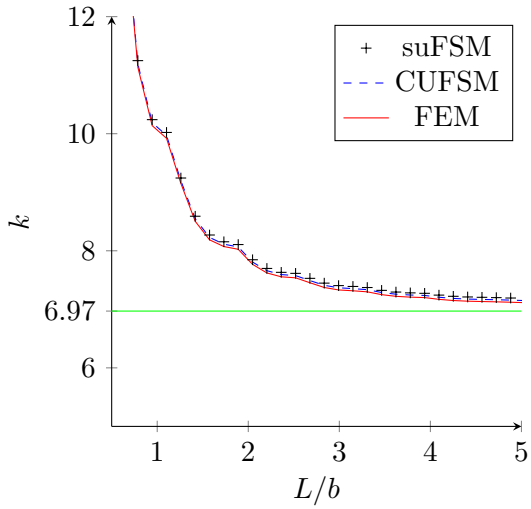


Figure 35: Buckling coefficients, C-C s-s.

From figure 7.3.2 it is clear that the results generated by the finite strip implementation developed in section 6 are in agreement with the results from the Finite Element model as well as CUFSM. This provides us with confidence that the output is sufficiently accurate and may be used as input into a Direct Strength design.

7.3.3 Discussion

In this section, the output from the finite strip implementation was verified for buckling solutions. This was done by determining the plate buckling coefficient for a 63.5mm wide 1.27mm thick plate with varying end conditions. Buckling curves were generated for length to width (L/b) ratios ranging from less than 1 to 5 using the finite strip implementation developed in section 6 and compared to values obtained from the commonly known CUFSM and FEM models. The results have shown that the finite strip implementation of section 6 delivers similar results when compared to CUFSM and FEM and as such the output may confidently be used as an input into a Direct Strength design.

8 Empirical Analysis

8.1 Design Example: C-Section with lips (DSM Design guide 2006)

8.1.1 Description

In the following examples, the design flexural strength for a fully braced beam and the design compressive strength for a continuously braced column are determined using the programme. The example problems are taken from AISI 2002 [4] and answers are compared to the DSM Design guide 2006 [3]. The values have been converted to metric units and rounded to 2 decimal places to compare SUFSM's answers with those given in [3]. Note that intermediate values in [3] are rounded to zero decimals.

Given:

1. Steel: $f_y = 379.21$ MPa
2. Section 9CS2.5x059 as shown on the right

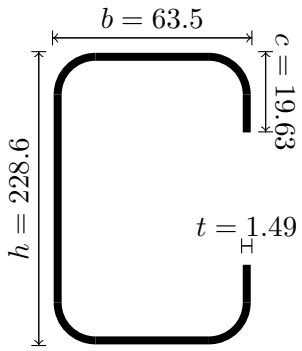


Figure 36: Channel section profile.

Required:

1. Flexural strength for a fully braced member
2. Flexural strength for $L = 1427.48$ mm
3. Compressive strength for a fully braced member
4. Compressive strength at $F_n = 256.83$ MPa

8.1.2 Flexural strength for a fully braced member.

The programme is started in DSM Design mode and a model is created with the profile shown in Figure 37. The model is given a length of $L = 2000$ mm.

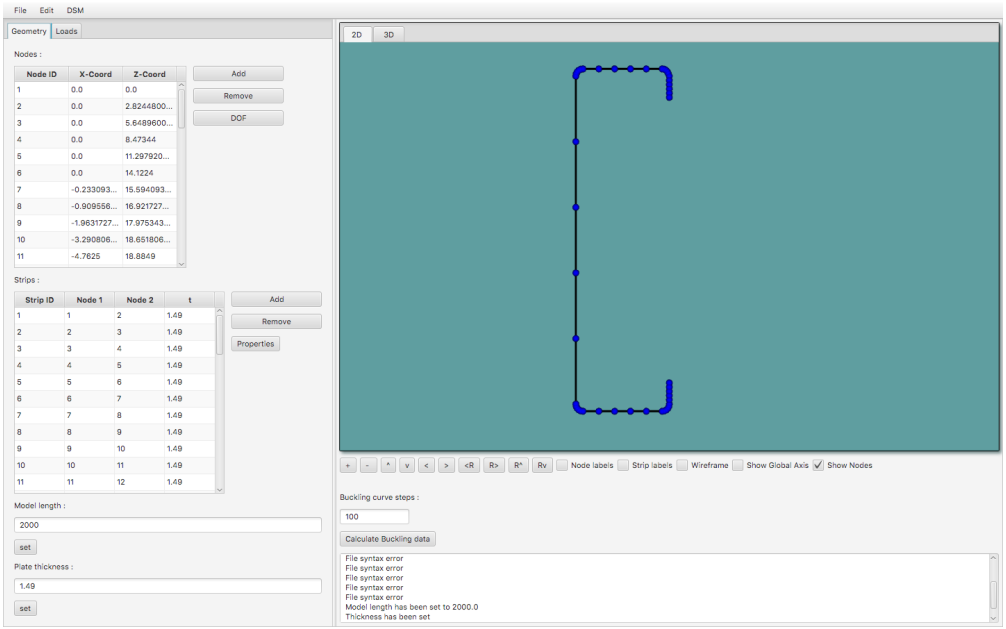


Figure 37: Model creation in SUFSM

The maximum centerline stress is set to 376.74 MPa and applied to the centreline of the cross section as indicated in Figure 38.

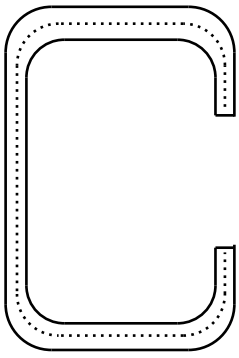


Figure 38: Position of centreline.

The value for the centreline stress is calculated by assuming a linear stress distribution from yield stress at the extreme fiber to zero at the neutral axis. A moment is applied to cause bending about the strong axis as shown in Figure 39.

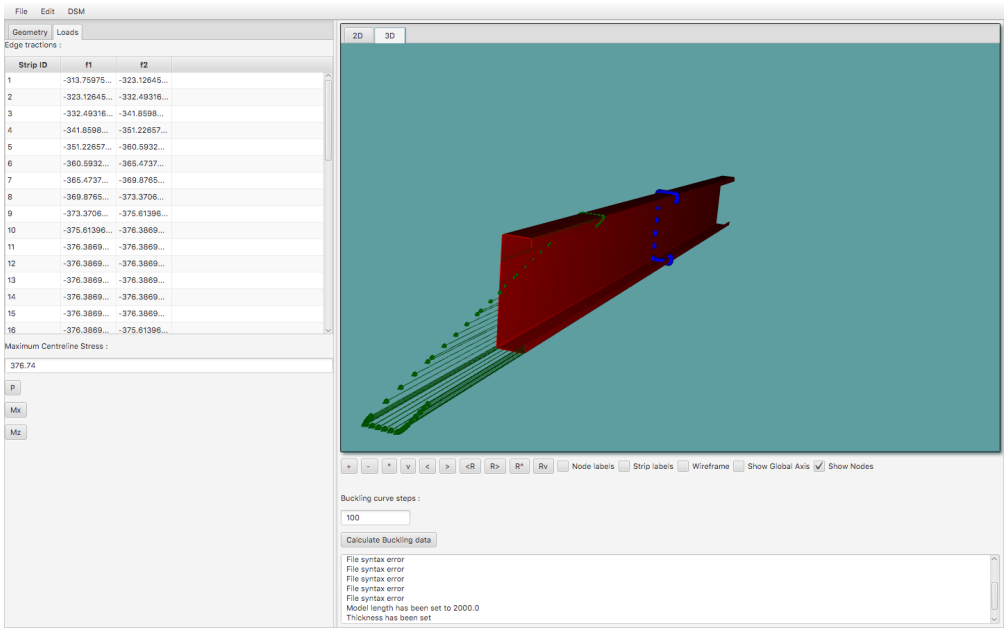


Figure 39: Edge traction applied in SUFSM

Finally, a buckling curve is generated with 400 plot points from which the local minima may be read off. Each local minimum represents a buckling moment and is given as a factor of the yield moment.

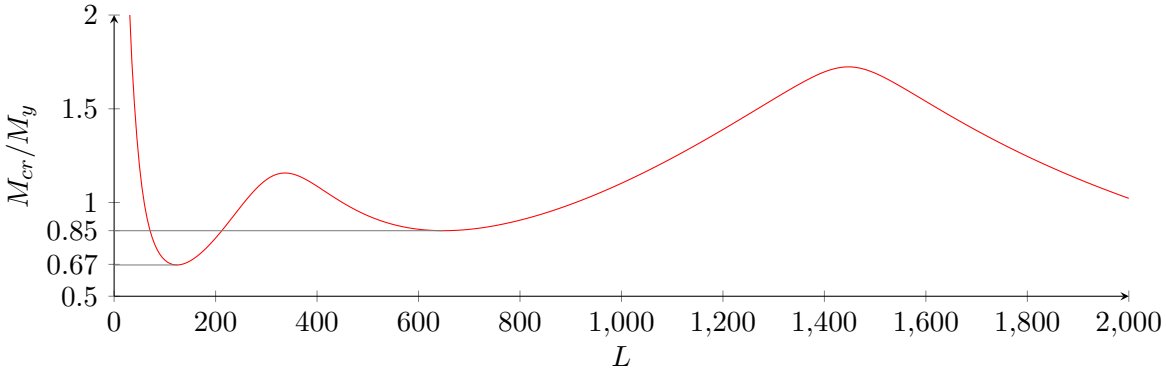


Figure 40: Signature curve generated using SUFSM

Values for critical-distortional M_{crd} and critical-local M_{crl} buckling moments from a finite strip analysis done with CUFSM are now compared to values obtained using SUFSM. The choice of global buckling moment factor M_{cre} is not important since the problem assumes that the member is fully braced.

Table 14: Critical moment comparison

Critical moment factor	CUFSM	SUFSM
M_{crl}	0.67	0.67
M_{crd}	0.85	0.85

The results are in exact agreement with those obtained from [3].

The local minima are identified from the buckling curve and the user simply selects the identified value, then using a right-click the value is stored in the program as either the local, distortional or global buckling factor.

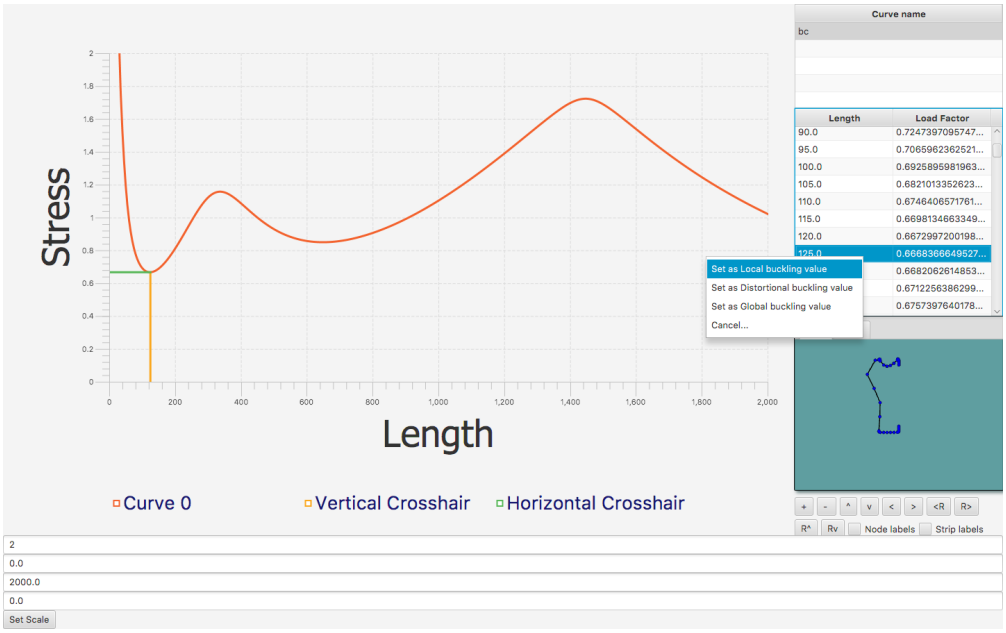


Figure 41: Buckling curve calculated using SUFSM

From the main window, the user selects DSM from the menu bar. A new window appears where the yield moment M_y for the section may be entered, but this value is already calculated by SUFSM. Note that this value might differ from the actual yield value since the cross-sectional properties of the model can differ slightly from those of the actual member.

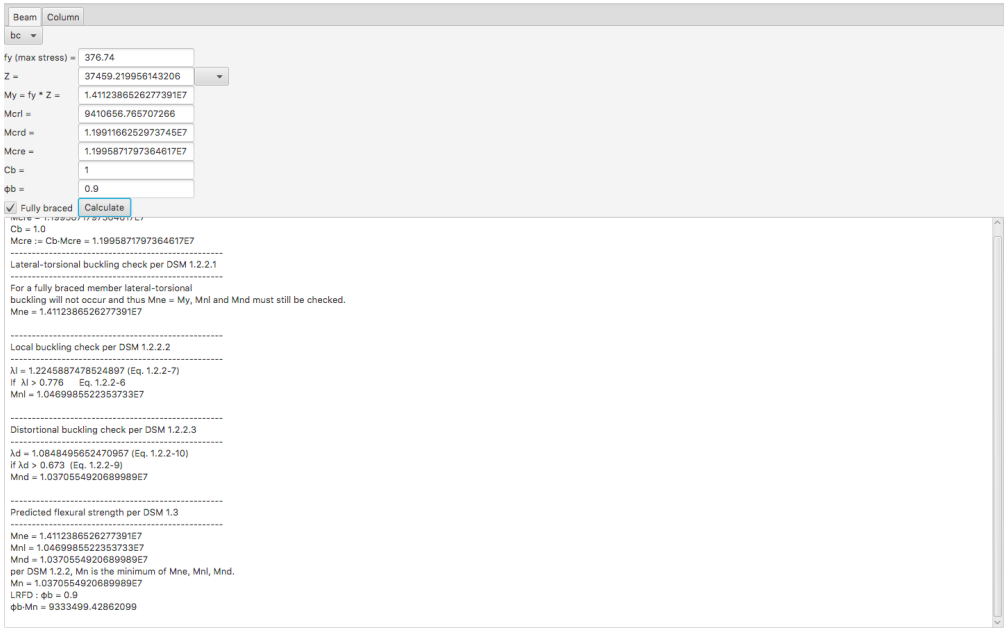


Figure 42: DSM design using SUFSM

With the option for fully braced selected, the nominal local, distortional and elastic buckling moments for the fully braced member are calculated by SUFSM. Now these values can be compared to the answers given by the DSM Design guide 2006 [3].

Table 15: Nominal moment comparison

Nominal moment	DSM Guide (KN.m)	SUFSM (KN.m)
M_{nl}	10.62	10.47
M_{nd}	10.51	10.37
M_{ne}	14.30	14.11

The results are in good agreement with those obtained from the DSM Design guide [3]. The small difference is expected due to the rounding made when converting from imperial to metric units and because the DSM Design guide [3] tends to round intermediate values. To obtain the design strength for the fully braced member, one only needs to apply a safety factor to the minimum of M_{nl}, M_{nd} and M_{ne} as per the DSM Design guide [3].

8.1.3 Flexural strength for $L = 1427.48$ mm

AISI (2002) [4] Example II-1 provides calculations for the design of a 4 span continuous beam. In the following, the flexural design strength of one of the spans is calculated. Namely, an interior span with length $L = 1427.48$ mm, and Cb = 1.67 (conservatively assumed as a linear moment diagram between the inflection point and support). The member is assumed to be unbraced.

The same procedure is followed as in 8.1.2 to determine the buckling curve while ensuring that M_{cre} is chosen at a length of 1427.48 mm.

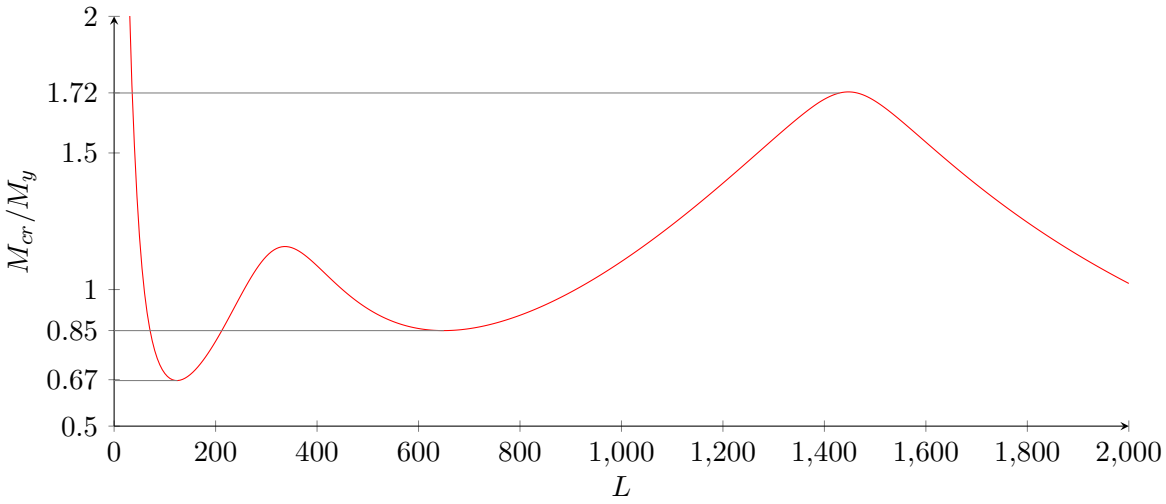


Figure 43: Signature curve generated using SUFSM

Values for critical-distortional M_{crd}, critical-local M_{crl} and critical-global M_{cre} buckling moments from a finite strip analysis done with CUFSM are compared to values obtained using SUFSM.

Table 16: Critical moment comparison

Critical moment factor	CUFSM	SUFSM
M_{crl}	0.67	0.67
M_{crd}	0.85	0.85
M_{cre}	1.73	1.72

The results are in good agreement with those obtained from [3]. The small deviation at the global end of the buckling curve is expected due to rounding when converting from imperial to metric units.

The same procedure is followed as in 8.1.2 to store the buckling factors. At the DSM window the option for fully braced should be deselected and the value for C_b should be set to 1.67 because a linear moment diagram is assumed between the inflection point and the suppport. Values for M_{nl}, M_{nd} and

M_{ne} according to [3] are now compared to values obtained using the program.

Table 17: Nominal moment comparison

Nominal moment	DSM Guide (KN.m)	Program (KN.m)
M_{nl}	10.62	10.47
M_{nd}	10.51	10.37
M_{ne}	14.30	14.11

The results are in good agreement with those obtained from [3]. As stated previously the small difference is expected due to the rounding made when converting from imperial to metric units and because the DSM Design guide [3] tends to round intermediate values. To obtain the design strength for the fully braced member, one only needs to apply a safety factor to the minimum of M_{nl}, M_{nd} and M_{ne} as per the DSM Design guide [3].

8.1.4 Compressive strength for a fully braced member

In AISI [4] Example I-8, the effective cross section properties of a C-section with lips is calculated. This example uses the same C-section and calculates the compressive strength for a fully braced member and compares the answers to the DSM Design guide [3]. As in 8.1.2, SUFSM is started in DSM Design mode and a model is created with the profile shown in Figure 36. The model is given a length of $L = 2000$ mm. The maximum centerline stress is set to 379.21 MPa and an axial load is applied to cause a stress of 379.21 MPa on the column face.

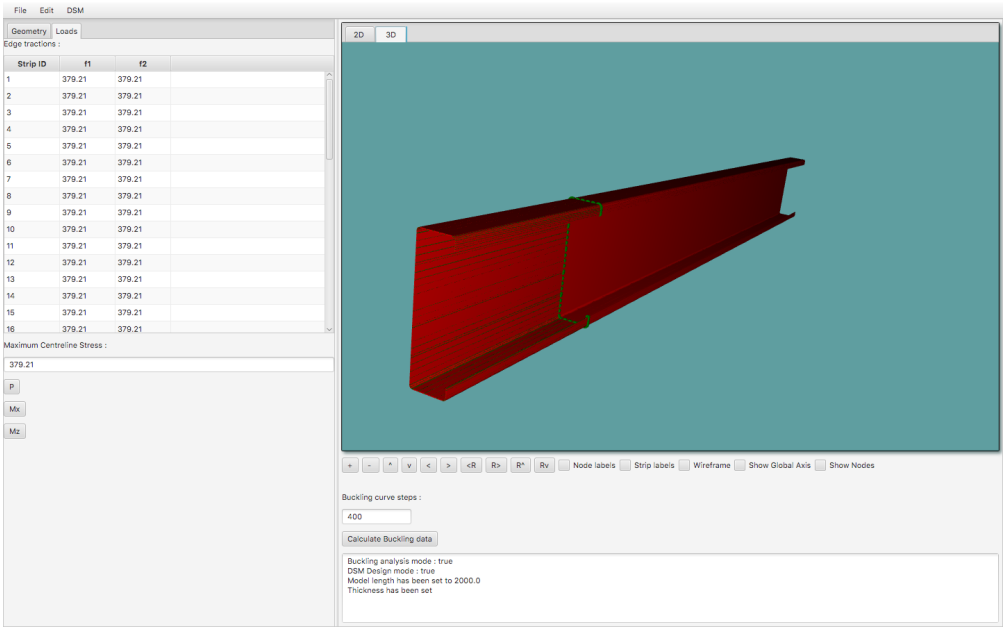


Figure 44: Profile creation using SUFSM

Finally a buckling curve is generated with 400 plot points from which the local minima and inflection point, may be read off. Each local minimum represents a buckling load and is given as a factor of the yield load.

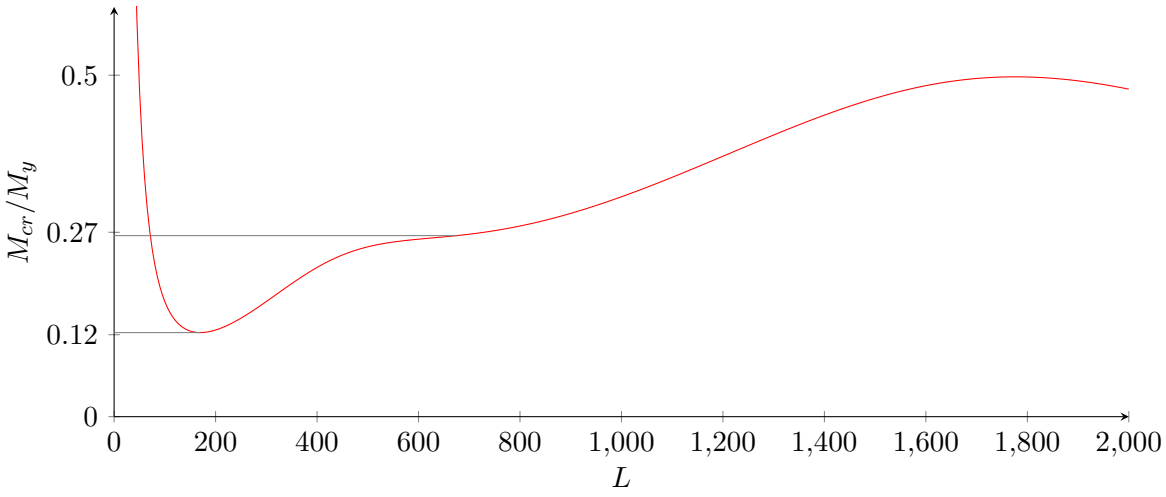


Figure 45: Signature curve generated using SUFSM

Values for critical-distortional P_{crd} and critical-local P_{crl} buckling loads from a finite strip analysis done with CUFSM are now compared to values obtained using SUFSM.

Table 18: Critical load comparison

Critical load factor	CUFSM	SUFSM
P_{crl}	0.12	0.12
P_{crd}	0.27	0.27

The results are in good agreement with those obtained from [3]. The local minima are identified from the buckling curve and the user simply selects the identified value, then using a right-click the value is stored in the program as either the local, distortional or global buckling factor.

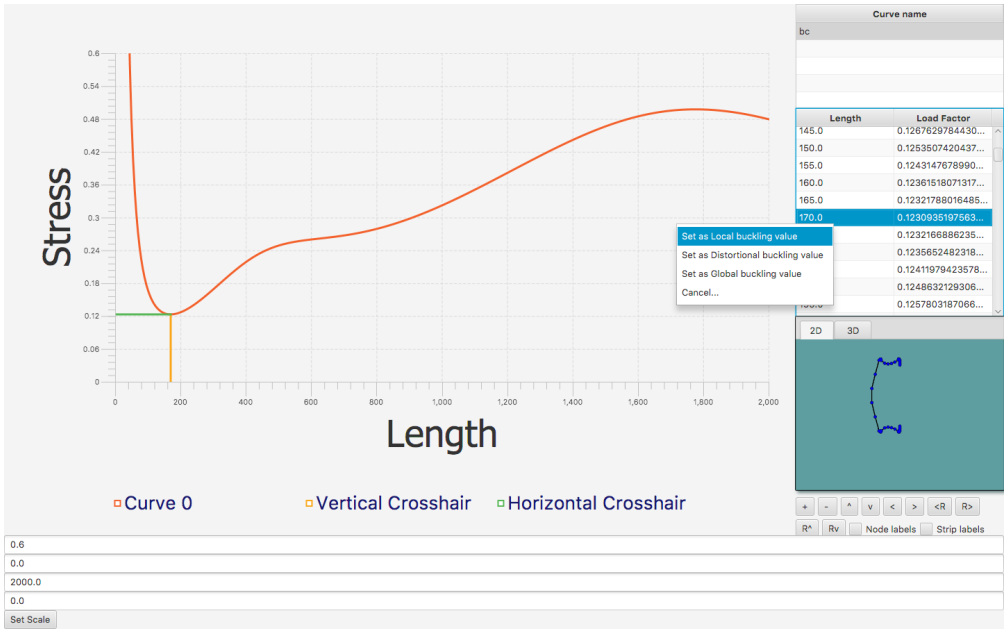


Figure 46: Buckling curve calculated using SUFSM

From the main window, the user selects DSM from the menu bar. A new window appears with two tabs at the top. The "column" tab should be selected. Now the yield load P_y for the section may be entered. Note that the value for P_y determined by the programme is not used, rather the section properties for the actual cross-section are calculated independently and P_y determined from there. This yields a better result because the straight line model provides a bad estimation of cross-sectional area.

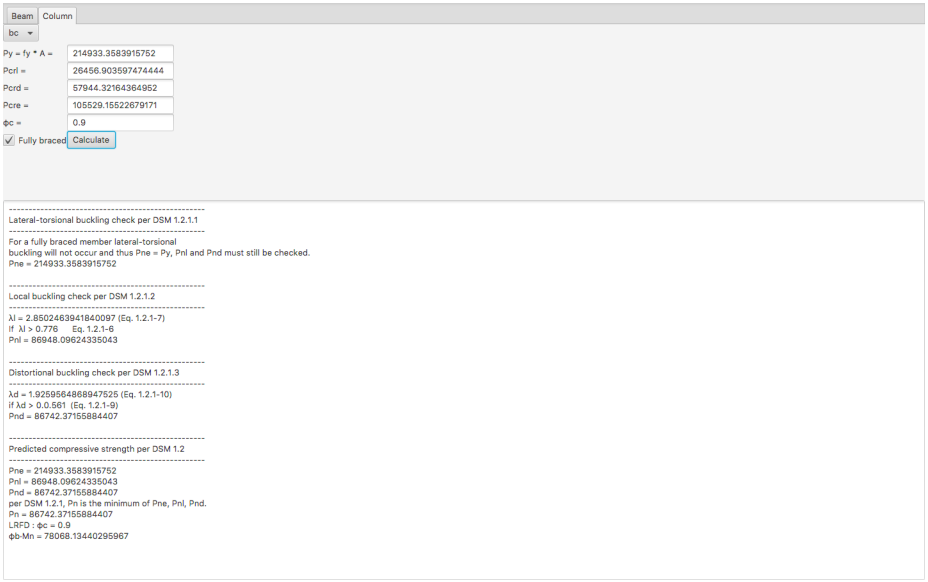


Figure 47: DSM design using SUFSM

With the option for fully braced selected, the nominal local, distortional and elastic buckling loads for the fully braced member are calculated by the programme. Now these values can be compared to the answers given by the DSM Design guide 2006 [3].

Table 19: Nominal load comparison

Nominal moment	DSM Guide (KN)	SUFSM (KN)
P_{nl}	86.29	87.33
P_{nd}	87.19	87.01
P_{ne}	215.38	215.38

The results are in good agreement with those obtained from [3]. To obtain the design compressive strength for the fully braced member, one only needs to apply a safety factor to the minimum of P_{nl}, P_{nd} and P_{ne} as per [3].

8.1.5 Compressive strength at $F_n = 256.83$ MPa

In AISI Example III-1 [4] this 9CS2.5x059 is examined as a 6.1m long beam-column. The section is simply-supported at its ends and fully braced against lateral and torsional buckling. The compressive design strength of that beam-column is to be determined [3].

As in 8.1.2, the programme is started in DSM Design mode and a model is created with the profile shown in Figure 36. The model is given a length of $L = 2000$ mm. The restriction to only allow buckling about the strong axis could be imposed on the finite strip model itself, however, one could simply calculate the elastic buckling stress using closed-form formulas as in AISI (2002) Example III-1 [4]. Following the procedure in AISI [4] yields $P_{cre} = 231.53$ kN. The maximum centerline stress is set to 379.21 MPa and an axial load is applied to cause a stress of 379.21 MPa on the column face. Finally, a buckling curve is generated with 400 plot points from which the local minimum and the inflection point may be read off.

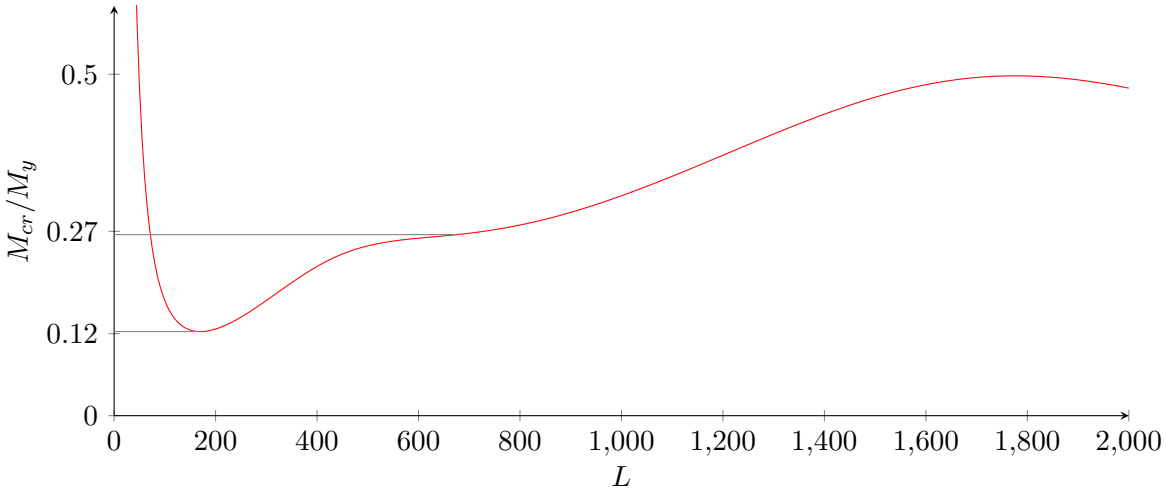


Figure 48: Signature curve generated using SUFSM

Each local minimum represents a buckling load and is given as a factor of the yield load. Values for critical-distortional P_{crd} and critical-local P_{crl} buckling loads from a finite strip analysis done with CUFSM are now compared to values obtained using SUFSM.

Table 20: Critical load comparison

Critical load factor	CUFSM	SUFSM
P_{crl}	0.12	0.12
P_{crd}	0.27	0.27

The results are in good agreement with those obtained from the DSM Design guide [3].

Using the minima from Table 20 as well as the calculated value of $P_{cre} = 231.53$ kN, the Direct Strength Method may be followed to determine the nominal-local P_{nl}, nominal-distortional P_{nd} and nominal-elastic P_{ne} buckling loads. These values are calculated by the programme and compared to the answers given by the DSM Design guide 2006 [3]. The manually calculated value for P_{cre} can be entered in the apropriate textfield on the 'DSM calcsheet' window as SUFSM will make use of the entered values for the DSM calculation.

Table 21: Nominal load comparison

Nominal moment	DSM Guide (KN)	SUFSM (KN)
P_{nl}	67.61	68.32
P_{nd}	87.19	87.01
P_{ne}	14.59	14.59

The results are in good agreement with those obtained from the DSM Guide [3]. The small difference is expected due to the rounding made when converting from imperial to metric units and because the DSM Guide [3] tends to round intermediate values. To obtain the design strength for the fully braced member, one only needs to apply a safety factor to the minimum of P_{nl}, P_{nd} and P_{ne} as per the DSM Design guide [3].

8.2 Discussion

In this section, the design flexural strength for a fully braced beam and the design compressive strength for a continuously braced column were determined using SUFSM. The example problems were taken from AISI 2002 [4] and answers were compared to the DSM Design guide 2006 [3]. The values have been converted to metric units and rounded to 2 decimal places to compare SUFSM's answers with those given in [3].

Comparison of the examples has shown that the results obtained from SUFSM are consistent with those obtained from CUFSM. in other words, SUFSM provides a means of accurately modelling the buckling behavior of the C-section used. Further, the Direct Strength calculations performed by SUFSM seem correct for the selected scenarios, as they are consistent with the results given by the DSM guide [3].

9 Conclusions

In this thesis, the fundamental Finite Strip theory has been reviewed to gain sufficient understanding in order to develop an implementation. A new FSM implementation called SUFSM was implemented in the object-oriented programming language JAVA™ making it more accessible, and the FSM technology itself more practicable and understandable. It demonstrates how the FSM can be used for static analysis of members as well as predicting buckling behaviour. Furthermore, SUFSM includes the ability to do Direct Strength based designs.

The FSM accurately models elastic buckling with minimal computational effort and has the ability to separate buckling modes for members with simply-supported boundary conditions. FSM, therefore, is a valuable tool in the design of cold-formed sections. Until now, designers who did not have access to MATLAB™ were limited to closed source implementations or had to create their own FSM implementation.

With the well renowned Finite Element Method so readily available, one could argue that the Finite Strip technology is obsolete, but in support of Finite Strip Technology this thesis has pointed out that the FEM does not meet the requirements for DSM. A comparison was also made between DSM and EWM to show that in some cases the DSM is advantageous to the EWM.

Throughout this thesis, two of the basic concepts behind the Finite Strip Method i.e. performing static analysis and buckling analysis were highlighted. It was also demonstrated how both of these concepts may be implemented in an object orientated programming language. JAVA was chosen because the source code is easy to understand and, when compiled, can execute on any machine with the JAVA runtime environment.

The fundamental implementation was tested to ensure that the output is reliable and may be used in design methods. The results were compared to existing FSM and FEM implementations. The results of the implementation were similar to those of CUFSM in terms of buckling analysis and similar to those of FEM in terms of static analysis. The implementation was also expanded to include strength calculations for members by means of the DSM. It was shown that the answers it provides are in good agreement with those given by the DSM Design Guide [3].

10 Recommendations

Researchers who do create their own implementation could find that they encounter accuracy issues with the use of the series functions described by Cheung [1]. Specifically when performing static analysis of members having boundary conditions other than simply supported and using a larger amount of longitudinal terms. The author found that the problem is caused by the use of the floating point primitive types in Java. In most cases, these floating point types cannot return an exact representation of a number. This behavior can be seen by setting a floating point type such as a double equal to zero then adding 5.8 and then adding 5.6. The answer should be 11.4 but the floating point type will return 11.39999... The problem is exaggerated by the fact that some terms in the series function are small relative to others. The author found that by using JAVA's built-in BigDecimal class that provides an accurate representation of the terms in the series function, the problem is solved.

The SUFSM software is in an early stage and still requires much refinement. As DSM is improved, so should the software be updated. Specifically, implementation of the constrained Finite Strip Method (cFSM) would be the logical next step as to enable modal identification of end conditions other than simply supported. Unfortunately, the Direct Strength Method is currently only applicable to simply supported members, the extension to other boundary conditions would be beneficial.

In literature, signature curves are usually graphically represented on a log-scaled x-axis. Currently, the JAVA FX libraries responsible for drawing the signature curves in SUFSM only support linear graphing. To make the results visually more consistent with what is encountered in literature, the author recommends that a custom API is used to draw the signature curves.

SUFSM's buckling analysis time can be greatly reduced by implementing the trigonometric series functions described by Bradford [8] and also used by CUFSM. But, as stated in this thesis care must be taken to only use these functions for buckling problems and not static analysis.

References

[1] Cheung Y.K. 1976. *Finite Strip Method in Structural Analysis.* Oxford: Pergamon Press.

[2] Li Z. 2011. *Finite Strip Modelling of Thin-walled Members.* Thesis, Ph.D. Johns Hopkins University.

[3] American Iron and Steel Institute. 2006 *Direct Strength Method (DSM) Design Guide..*

[4] American Iron and Steel Institute. 2003 *AISI Manual Cold-Formed Steel Design 2002 Edition.* AISI-Specifications for the Design of Cold-Formed Steel Structural Members. Paper 130.

[5] Schafer B.W. and Peköz T. 1998, *Direct strength prediction of cold-formed steel members using numerical elastic buckling solutions.* International Specialty Conference on Cold-Formed Steel Structures. Paper 4.

[6] SANS. *The structural use of steel. Part 2: Cold-formed steel structures.* SANS 10162-2:2011.

[7] Schafer B.W., 2008, *Review: The Direct Strength Method of cold-formed steel member design* Journal of Constructional Steel Research 64 p.766-778.

[8] Bradford, M.A. and M. Azhari, 1995, *Buckling of plates with different end conditions using the finite strip method.* Journal of Computers & Structures vol 56, p. 75-83.

[9] Freitas, M.S.R. Brandão, A.L.R. and Freitas, A.M.S., 2013, *Resistance factor calibration for cold-formed steel compression members*, Federal University of Ouro Preto.

[10] Moen, C.D., 2007, *Direct Strength Design of Cold-Formed Steel Members with Perforations*, The Johns Hopkins University.

[11] Ziemian R.D., 2010, *Guide to Stability Design Criteria for Metal Structures, Sixth Edition*, John Wiley & Sons, inc.

[12] MacDonald M. and Kulatunga M.P., 2013 ,*Finite Element Analysis of Cold-Formed Steel Structural Members with Perforations Subjected to Compression Loading*, Mechanics and Mechanical Engineering Vol. 17, No. 2 p. 127-139, Lodz University of Technology

[13] Von Karman T., Sechler E.E., Donnell L.H. 1932 *The strength of thin plates in compression.* The American Society of Mechanical Engineers Vol. 54.

[14] Winter G. 1948. *Performance of thin steel compression flanges.* IABSE Congress Report.

[15] American Iron and Steel Institute, 1946. *Specification for the Design of Light Guage Steel Structural Members.* AISI-Specifications for the Design of Cold-Formed Steel Structural Members. Paper 1.

[16] Landesmann A. and Camotim D., 2010, *On the Strength and DSM Design of Cold-formed Steel Columns Failing Distortionally Under Fire Conditions* , Federal University of Rio de Janeiro.

[17] Zienkiewicz O.C., Taylor R.L., Zhu J.Z., 2005, *The Finite Element Method: Its Basis and Fundamentals Sixth edition*, Elsevier.

[18] Li Z. , Schafer B.W., 2009, *Buckling Analysis of Cold-formed Steel Members with General Boundary Conditions Using CUFSM Conventional and Constrained Finite Strip Methods*, The Johns Hopkins University.

[19] Silvestre N., Camotim D., 2002, *First-order generalised beam theory for arbitrary orthotropic materials*, Thin-Walled Structures Vol 40 p. 755789. Elsevier.

[20] Schafer, B.W., 1997, *Cold-Formed Steel Behavior and Design: Analytical and Numerical Modeling of Elements and Members with Longitudinal Stiffeners*, Ph.D. Thesis. Cornell University.

[21] SAISC, 2016, *South African Steel Construction Handbook 8th Edition*, South African Institute of Steel Construction.